Modeling the Mechanical Behavior of Soft Active Materials

by

Jingkai Guo

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

July, 2018

Abstract

Soft active materials are capable of converting chemical or physical energy into mechanical work in response to various environmental stimuli such as pH, solvent, salt, heat, humidity, electric or magnetic field, and light. These materials are generally lightweight, inexpensive, fracture tolerant and pliable. They can be configured into various conceivable shapes and their properties can be tailored to suit a broad range of requirements. Because of these attractive characteristics, they have attracted considerable attention for biomimetic or bioinspired systems. The objective of this work is to develop models that are capable of accurately describing the physical-mechanical behavior of soft active materials. A common set of methods that can be applied to the study of various kinds of soft materials was developed. The methods covered different stages of the study, from material characterization to constitutive modeling, from simulation to experiments.

In this work, we focused on three kinds of soft active materials: dielectric elastomers, thermoresponsive gels and shape memory polymers. In the first part of this work, we investigated the temperature dependent viscoelastic behavior of dielectric elastomers and the effects of viscoelasticity on the electro-actuation behavior. We measured the viscoelas-

ABSTRACT

tic relaxation spectrum of VHB 4905 and applied the results to a discrete multi-process viscoelastic model. The model generally showed good quantitative agreements with experimental measurements in terms of both the pure-mechanical and electro-mechanical behaviors of the material. The model was able to qualitatively capture the dependence of the electric breakdown time on voltage and pre-stretch. In the second part of this work, we studied thermoresponsive bilayer plates with soft and stiff segments that exhibit bidirectional and biaxial curving. We investigated the mechanism underlying the deformation modes through finite element simulation and explored the effects of geometrical factors such as the aspect ratio and the segment spacing. In the final part of this work, we developed a constitutive model based on the effective temperature theory that is capable of describing strain hardening behavior in a thermodynamically consistent manner. The model incorporated two mechanisms: one represents the stretching and orientation of the polymer network, which leads to the development of a backstress; the other one represents the network relaxation, which accounts for the temperature and rate dependence of strain hardening. The model was implemented into finite element programs and was applied to simulate the thermomechanical behavior of polycarbonate. All of the model parameters were determined through standard thermomechanical tests. The simulation results showed good agreement with experiments and the temperature dependence, strain rate dependence and strain state dependence of hardening were quantitatively captured.

ABSTRACT

Primary Reader:

Thao D. (Vicky) Nguyen, Associate Professor, Department of Mechanical Engineering and Department of Materials Science and Engineering, Johns Hopkins University

Secondary Readers:

David H. Gracias, Professor, Department of Materials Science and Engineering and Department of Chemical and Biomolecular Engineering, Johns Hopkins University

Mark O. Robbins, Professor, Department of Physics and Astronomy, Johns Hopkins University

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisor, Professor Vicky Nguyen, who gave me this great opportunity to study abroad, to join Hopkins. When I first stepped on this adventure six years ago, I knew little about continuum mechanics and had little research experience. With her continuous patience and support, I made progresses step by step. I learnt a lot from her in the past six years, not only knowledge and skills, but also the attitude and passion of being a researcher. That will benefit me through my life.

I would like to thank Professor David Gracias and Professor Mark Robbins who serve as committee members of my thesis defense, for the collaboration and support you provided during my research, and for taking time to read this thesis. I would also like to thank Professor Michael Falk, Professor Jaafar El-Awady, Professor James Guest and Professor Margarita Herrera-Alonso who served as committee members of my qualifying exams, for their valuable suggestions and comments to my work.

I also want to thank all of my collaborators: Professor Harold Park from Boston University and Kailiang Ren on the project of dielectric elastomers; Professor David Gracias, Professor Rebecca Schulman, Tanvi Shroff, ChangKyu Yoon, Angelo Cangialosi and Qi

ACKNOWLEDGMENTS

Huang on the project of hydrogels; Professor Mark Robbins, Professor Sung Hoon Kang and Lichen Fang on the project of shape memory polymers. We came from very different backgrounds, but we were able to collaborate effectively and achieved something together.

I would also like to thank my labmates in Nguyen's group. I always enjoy the atmosphere of the group and it was great working with Rui Xiao, Baptiste Coudrillier, Theresa Koys Tonge, Barbara Murienne, Aurlie Azoug, Bahram Notghi, Dan Midgett, Michelle Chen, Tracy Liu, Hai Dong, Zheng Jia, Jiayu Liu, Tracy Ling, Zheliang Wang and Cristina Martin Linares. Special thanks to Rui Xiao, who pioneered the research in shape memory polymers and hydrogels, for the mentorship and advice I received from him in my first couple of years at Hopkins. I also want to thank undergraduate research assistants who worked with me, Jonathan Wang and Zhezhi Chen, for their assistance in experiment setup. And I would like to thank the staff members in Department of Mechanical Engineering and extend my gratitude to the whole Mechanics and Materials community at Hopkins. They were always friendly and willing to help.

I am grateful for my funding sources, which include the NSF (grant number CMMI-1130358) and the Army Research Office Grant (grant number W911NF-15-1-0490).

Finally, I would like to thank my family, particularly my parents. Their unconditional love, support and encouragement is the best company through this long journey.

Dedication

This thesis is dedicated to my family, particularly my grandma.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter provides a brief review for soft active materials. Particular attention is given to the three kinds of materials that were studied in this thesis: dielectric elastomers, thermoresponsive gels and shape memory polymers. For each kind of material, the introduction describes the physical, mechanical and thermodynamical properties as well as representative applications. The introduction provides background information for the discussions in following chapters. The chapter concludes with the objectives of this thesis.

1.1 Soft Active Materials and their Applications

Soft active materials are capable of transforming various forms of energy to mechanical work in response to environmental stimuli such as pH, heat, humidity, solvent, salt, electric or magnetic field, and light. These materials are generally inexpensive, lightweight and

their properties can be tailored based on requirements of applications. These attractive features give them great potential for a wide range of applications, such as biomimetic systems inspired from living organisms [2, 20]. A variety of soft materials and their corresponding stimuli are summarized in Table 1.1.

Materials	Stimuli
Gel	pH, salt, solvent, heat, light, electric and magnetic fields
Conductive polymer (CP)	electric field and humidity
Shape memory polymer (SMP)	heat, light, electricity, pH and solvent
Elastomer	electric field and light
Carbon nanotube (CNT)	electric field etc.
Biomaterial	ATP etc.

Table 1.1: Soft active materials and their corresponding stimuli.

The term ”soft” is a relative concept. Fig. 1.1 summarizes the Young's modulus of various materials [4]. Soft materials describe a broad class of substances, from glassy polymers with a Young's modulus of 1-3 gigapascals to hydrogels with moduli on the order of kilopascals [21]. A more precise definition for soft materials was given by De Gennes [22], who characterized soft matter as a class of materials that give a ”large response to small perturbations” instead of being physically soft. This definition is more clear and straightforward in the context of this thesis. Any material that exhibits large strains in response to external stimulus can be described as ”soft”. Soft materials generally have some common features in terms of microstructures. The materials are held together by weak intermolecular interactions instead of atomic bonds, which makes the structure easily affected by energies in the order of k_BT, where k_B is the Boltzmann constant and T is

temperature. The microstructure is in a relatively disordered state, with no ordering or only short range ordering in the arrangement of molecules, compared to the long range ordering in crystals [21].

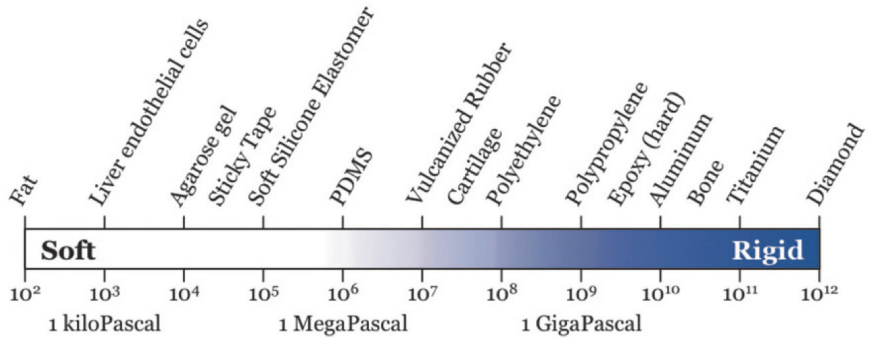


Figure 1.1: Young's modulus for various materials, reprinted with permission from Mary Ann Liebert, Inc. [4].

Active materials, also referred to as smart materials, intelligent materials or adaptive materials, describe those that are capable of converting energy among multiple physical states [5]. A physical state can be described by a set of state variables. For example the thermal state can be described by state variables of temperature and entropy. Other examples are summarized in Fig. 1.2 [5]. Active materials exhibit coupling between multiple physical states, which means the state variables in one physical state has an impact on the state variables of another physical state. For example, electric charge accumulate in piezoelectric materials in response to applied mechanical stress, which is known as electromechanical coupling. Based on this idea, an important method for modeling soft active materials, called the state variable approach, can be developed. This approach involves basic balance laws, state variables in different physical states, and constitutive equations

that relate state variables together. The first law of thermodynamics, which states that the change in internal energy of a system is equal to the heat added to the system minus the work done by the system, brings together state variables from different physical states. For example, the mechanical work given by the product of stress and strain, and the electrical work given by the product of electric field and electric displacement, both result in work being done on a body and an increase in internal energy. This state variable approach is applied and discussed in detail in the following chapters of this thesis.

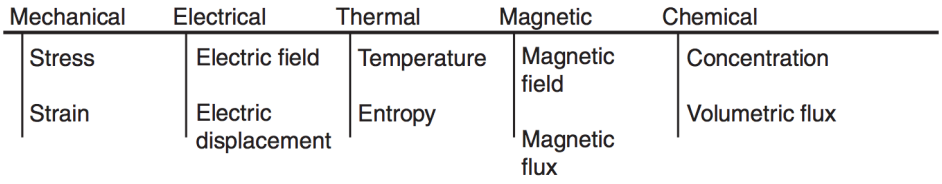
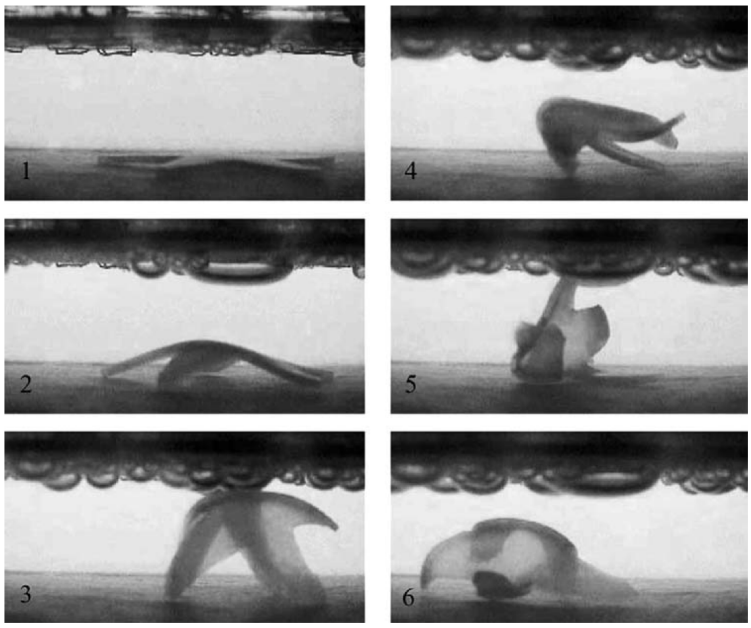


Figure 1.2: Physical states and associated state variables, reprinted with permission from Wiley [5].

A number of soft actuators have been made from synthetic or natural materials. These soft actuators were further developed to lightweight, inexpensive and efficient artificial muscles and soft robots for practical applications (2, 23–33). Traditional robots are made from rigid materials, which generally makes them bio-incompatible because of the giant mismatch in stiffness between the robot structure and the biological tissues. In addition, the rigid structure may limit the robots' capability of interacting with the surrounding environment under some circumstances, for example, performing tasks in congested environments. In contrast, biological organisms with soft structures can adapt to environmental changes and generate sophisticated motions. Examples include an elephant trunk and octopus arm

etc. (34–37) Inspired by these natural structures (38–45), researchers have designed various soft robots for practical applications. Unlike rigid robots that have limited number of degrees of freedom granted by actuators installed at multiple joints, soft robots have actuators distributed over the whole body, which gives them theoretically infinite number of degree of freedoms and configurations [46]. Moreover, soft robots are capable of squeezing into confined spaces and exhibit large strain and biocompatibility [4]. Examples of several soft robots are shown in Fig. 1.3 [4, 46].



(a)

(b)

Figure 1.3: Examples of soft robots (a) A starfish gel robot that turns over, reprinted with permission from Elsevier [6]. (b) Soft field robot for military reconnaissance, natural disaster relief, and pipe inspection, reprinted with permission from Mary Ann Liebert, Inc. [4].

In the scope of this work, we focused on three kinds of soft materials: dielectric elas-

tomers, thermo-responsive gels and shape memory polymers. They are introduced respectively in the following sections.

1.2 Dielectric Elastomers

1.2.1 Dielectric Elastomers as High-Performance Electroactive Polymers

Electroactive polymers (EAPs) can deform when exposed to electronic fields. As a representative category of soft active materials, they have exhibited great potential in widespread applications. There are two kinds of activation mechanism for EAPs, electrical activation driven by electrical fields and ionic activation driven by the motion of ions. Among various EAPs, dielectric elastomers, as a kind of electronic EAP, are closest to animal muscles [47]. Compared to other active materials, dielectric elastomers have unique attractive features such as fast response and good electro-mechanical converting efficiency [1]. Table 1.2 compares muscles with several kinds of active materials [1]. As shown in the table, although their cycle of life is relatively short, dielectric elastomers match or overmatch muscles in terms of stress and strain level, work density and continuous power [31, 48]. Some representative applications and corresponding advantages of dielectric elastomers are summarized in Table 1.3 [2].

	Mammalian Skeletal Muscle	**Dielectric Elastomers**	**Relaxor Ferroelectric Polymers**	**Conducting Polymers**	**NiTi Shape Memory Alloys**
Strain (%)	20-40	10 to >100	3-10	2-12	1-8
Stress (MPa)	0.1-0.35	0.1-2 typical, 3-9 maximum	20-45	1-100	200
Work density (kJ/m^3)	8-40	10-150 typical, 3400 maximum	1000	70-100	>1000
Density (kg/m^3)	1037	1000	2000	1300	6450
Strain rate (%/s)	>50	450 in VHB, 34000 in silicone	>2000 at 0.1% strain	1-12	300
Continuous power (W/kg)	50-280	500	>1000	150	>1000
Electro-mechanical coupling (%)	<40 efficiency	30 typical, 90 maximum	10-40	<10	<5
Cycle life	$>10^9$	10^6 at 50% strain	-	800000 best to date	300 at 5%, 10^7 at 0.5%
Modulus (MPa)	$<10-60$	0.1-3	400-1200	500	20000-80000
Voltage (V)	<1	>1000	1000	2	Low

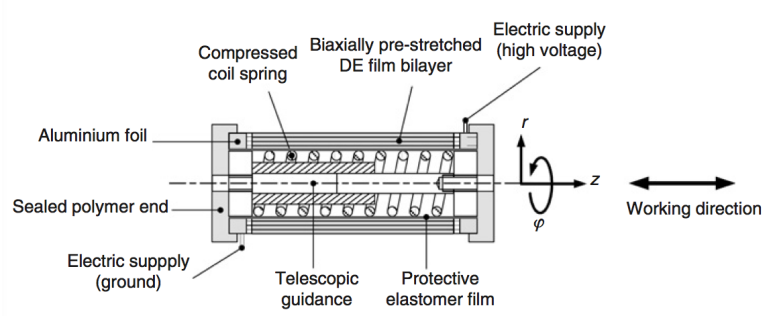
Table 1.2: Actuators comparisons, data adapted from [1].

Application	**Existing Actuator Technology**	**Potential Advantages of DEs**
Motors	Electromagnetics	Higher power density, lower-speed operation (eliminates gearing); low-cost materials
Linear Actuators	Electromagnetics, ceramic piezoelectrics (small stroke)	Higher energy density, higher power density, greater efficiency at low speeds; lower-cost materials and production; greater variety of shapes and sizes
Loudspeakers	Electromagnetic, ceramic, electrostatic (air gap)	Lighter weight; more flexibility in design shape (e.g., conformal, flat), higher power output; low-cost materials
Elastomer	Electromagnetics	Higher power density, lower speed operation, novel designs; lower cost materials
Robotic Actuators	Electromagnetic motors	Higher energy and power densities; ability to operate in direct drive; greater simplicity; ability to be configured in novel shapes more easily (e.g., snakelike robots); lower cost; ability to combine actuation, sensing, and structure in a single material; ability to match the passive behavior of natural muscle for biomimetics

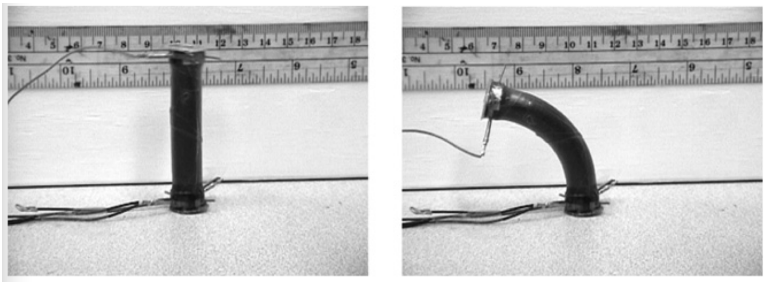
MEMS	Electrostatics	Higher energy and power density; simpler design; imperviousness to dust and other contaminants; ability to combine actuation and structure

Table 1.3: Representative applications for dielectric elastomers, table adapted from [2].

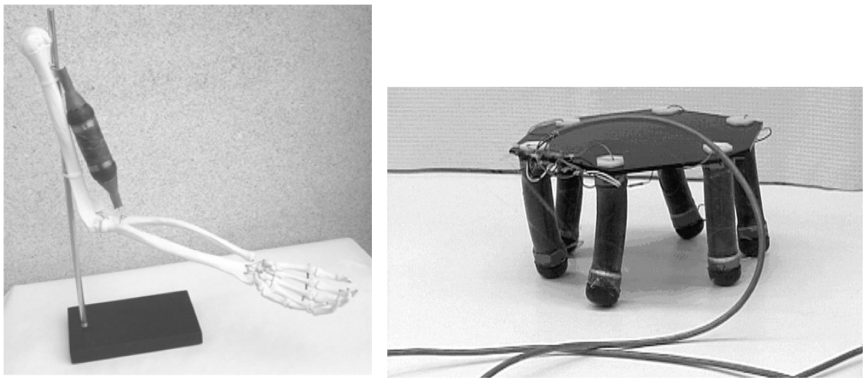
Dielectric elastomers are capable of reversible conversion between electrical energy and mechanical energy. Therefore, they can be developed for sensors in which the mechanical work is converted to electrical signals, or be developed to actuators that convert electrical energy to mechanical work. One of the intriguing features of DE based actuators is that they can achieve three dimensional motions with one dimensional configurations, which makes them simple to manufacture and mechanically robust [49, 50]. This can be realized by proper patterning of the compliant electrodes. One representative example is the spring roll actuator, as shown in Fig. 1.4. A schematic of the design of a rolled DE actuator is shown in Fig. 1.4(a) and a spring roll that can extend in the axial direction and bend is shown in Fig. 1.4(b). Fig. 1.4(c)-1.4(d) show a DE artificial muscle and a six-legged robot as examples of applications of the roll actuators [1].



(a)



(b)



(c) (d)

Figure 1.4: Spring roll DE actuators and their applications, images adapted from [1], (a) Schematic of the design of rolled DE actuators, (b) A spring roll actuator exhibiting elongation and bending, (c) Rolled actuator on a full-size skeletal arm model, (d) A soft robot using spring rolls as legs.

1.2.2 Thermodynamics of Dielectric Elastomers

The thermodynamic theory of dielectric elastomers was developed by Suo [7] and is briefly summarized below. From the first law of thermodynamics, when the material subjected to an applied force P and an applied voltage Φ, the change in the Helmholtz free energy δF equals the sum of the mechanical work and electrical work,

$$\delta F = P\delta l + \Phi\delta Q, \tag{1.1}$$

where δl is a small variation in displacement and δQ is a small variation in amount of charge. The Helmholtz free energy of the material is a function of the two independent state variables (l, Q), $F = F(l, Q)$. Therefore,

$$\delta F = \frac{\partial F(l,Q)}{\partial l}\delta l + \frac{\partial F(l,Q)}{\partial Q}\delta Q. \tag{1.2}$$

From eq. (1.1) and eq. (1.2),

$$\left(\frac{\partial F(l,Q)}{\partial l} - P\right)\delta l + \left(\frac{\partial F(l,Q)}{\partial Q} - \Phi\right)\delta Q = 0. \tag{1.3}$$

For arbitrary variations δl and δQ, eq. (1.3) must hold. Consequently, the two coefficients on the left hand side equal zero, giving,

$$\begin{aligned} P &= \frac{\partial F(l,Q)}{\partial l}, \\ \Phi &= \frac{\partial F(l,Q)}{\partial Q}. \end{aligned} \tag{1.4}$$

The above equations can be applied to a parallel-plate capacitor as shown in Fig. 1.5. From the elementary fact that the amount of charge on either electrode is linear with voltage,

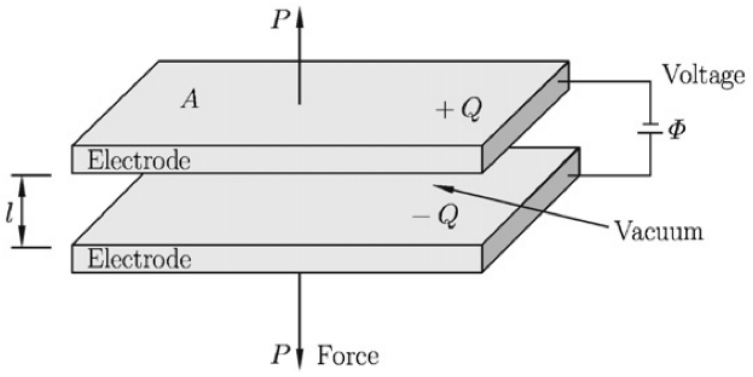


Figure 1.5: A parallel-plate capacitor subjected to force P and voltage Φ, figure adapted from [7].

$$\Phi = \frac{lQ}{\varepsilon_0 A}, \tag{1.5}$$

where ε_0 is the permittivity of the vacuum and A is the area of the electrodes. Substituting eq. (1.5) to eq. (1.4) and integrating gives,

$$F(l,Q) = \frac{lQ^2}{2\varepsilon_0 A}. \tag{1.6}$$

Inserting eq. (1.6) into eq. (1.4) gives

$$P = \frac{Q^2}{2\varepsilon_0 A}. \tag{1.7}$$

Defining electric field $E = \Phi/l$ and stress $\sigma = P/A$, and rewriting eq. (1.7) as,

$$\sigma = \frac{1}{2}\varepsilon_0 E^2. \tag{1.8}$$

gives the stress needed to balance the electrostatic attraction and this stress is known as the Maxwell stress. For ideal dielectric elastomers [7], the Maxwell stress shares the similar quadratic form with eq. (1.8), replacing the permittivity of vacuum ε_0 to the permittivity of the material ε,

$$\sigma = \frac{1}{2}\varepsilon E^2. \tag{1.9}$$

Fig. 1.6 shows the stress vs. electric field data from experiments and quadratic fits from eq. (1.9) [8].

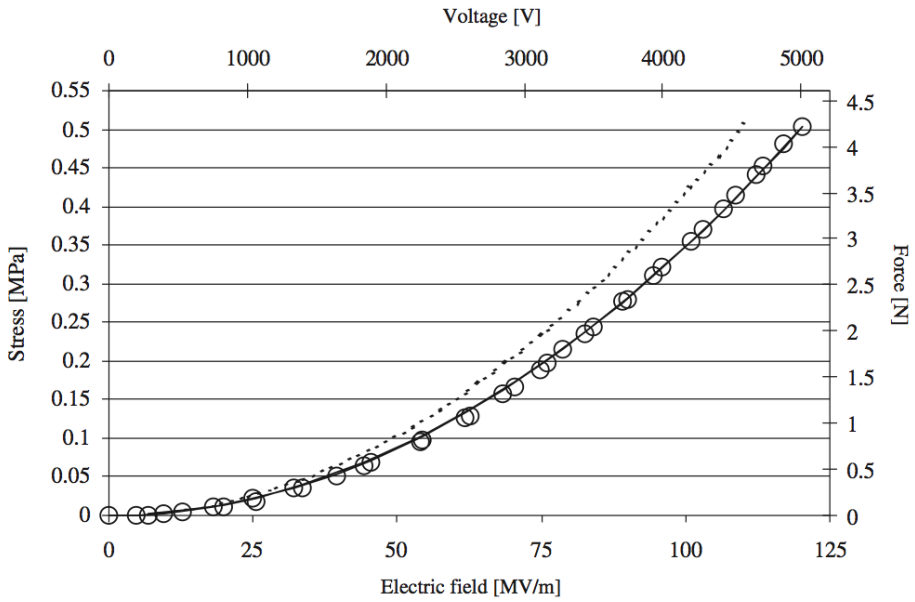


Figure 1.6: Stress vs. electric field data from experiments for VHB 4910 and quadratic fits from eq. (1.9) with $\varepsilon = 4.7\varepsilon_0$, reprinted with permission from SAGE [8].

1.3 Thermo-responsive Gels

Stimuli responsive gels have been widely applied in the development of multi-functional biomimetic structures [51]. Among various kinds of stimuli responsive gels, thermo-responsive gels are particularly intriguing, because temperature change widely exists in nature and temperature as an external stimulus can be controlled easily and accurately. Moreover, the phase transition of thermo-responsive gels in aqueous solutions is attractive for biomedical applications, such as drug delivery and switchable synthetic cell culture surfaces (52–54).

The behavior of a polymer in aqueous solution is the consequence of competition between like and unlike interactions among polymer molecules and solution molecules. For

thermo-responsive gels, a phase transition occurs in response to a temperature change, where the polymer-solution system transforms from a homogeneous phase to a two-phase state consisting of a polymer concentrated phase and a dilute phase. In this process, the clear solution becomes cloudy at a certain temperature known as the cloud point (CP) because of the formation of droplets with high concentrations of polymers [10]. When the phase separation happens above a threshold temperature, the transition is known as a lower critical solution temperature (LCST) transition; when it happens below a threshold temperature, the transition is known as an upper critical solution temperature (UCST) transition. Different types of phase transition are illustrated in Fig. 1.7 [9], namely the LCST transition, UCST transition and closed loop coexistence of LCST and UCST transitions. The schematically drawn binodal curves represent the equilibrium concentration of the two phases in the phase-separated state. The LCST is defined as the lowest temperature of the binodal curve (Fig. 1.7 (a)) while the UCST is defined as the highest temperature of the binodal curve (Fig. 1.7 (b)). Closed loop coexistence has also been reported for a small number of polymers (Fig. 1.7 (c)).

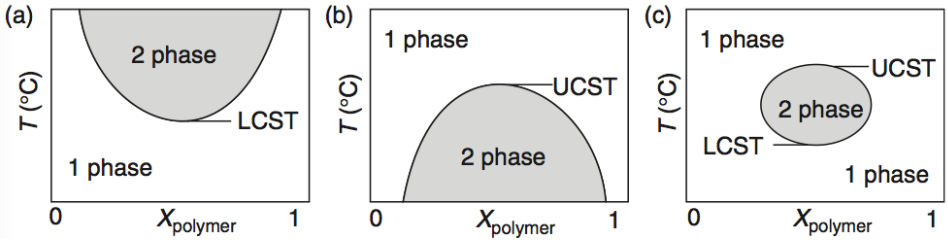


Figure 1.7: Schematic representation of the polymer phase diagrams for polymers exhibiting (a) LCST behavior, (b) UCST behavior and (c) closed loop coexistence, figures reprinted from [9].

In this thesis, we focused on the LCST transition. Gels with an LCST exhibit phase separation above some critical temperature. The hydrophilic polymer chains become hydrophobic, resulting in a dehydrated polymer rich phase. Structures of poly(N-isopropylacrylamide) (pNIPAM) and poly(N-vinyl caprolactam) (pVCL), two kinds of gels that undergo an LCST transition, are shown in Fig. 1.8 [9]. From a thermodynamic view, the Gibbs free energy ($\Delta G = \Delta H - T\Delta S$) of mixing the polymer with water is negative at low temperature, but becomes positive at high temperature. Such behavior is possible only if both of the following two conditions are met: a) the enthalpy of dissolution ΔH is negative, which means the bonding between polymer chains and water molecules is favorable; b) the entropy contribution ΔS is also negative, which means that entropy decreases for hydration. Upon increasing the temperature, the entropy term ($T\Delta S$) becomes dominant, leading to a positive Gibbs free energy of mixing and consequently phase separation. The LCST transition is fully reversible. In general, the LCST strongly depends on the polymer structure. For example, introducing hydrophilic end-groups or more hydrophilic (co)monomers increases the LCST. The simplest and most convenient method of determining the LCST is by measuring the cloud point. A typical cloud point curve is illustrated in Fig. 1.9 [10]. Another method is by using differential scanning calorimetry (DSC). Heskins and Guillet [55] first reported that an endothermic overshoot can be observed at the LCST upon heating aqueous solutions of pNIPAM. As demonstrated in Fig. 1.9, the transition temperature detected by DSC is in good agreement with results from cloud point measurements.

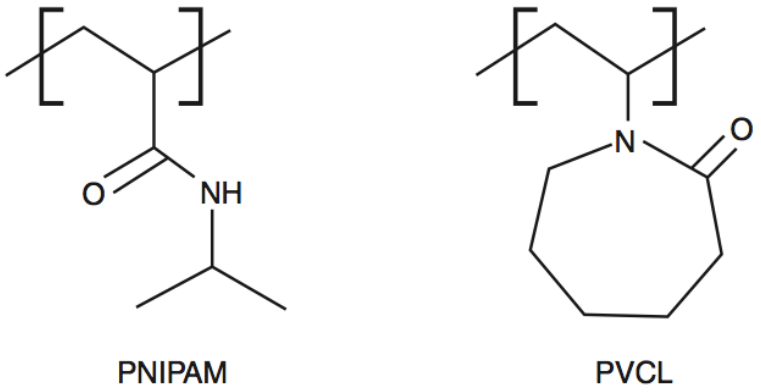


Figure 1.8: Structures of poly(N-isopropylacrylamide) (PNIPAM) and poly(N-vinyl caprolactam) (PVCL), figures reprinted from [9].

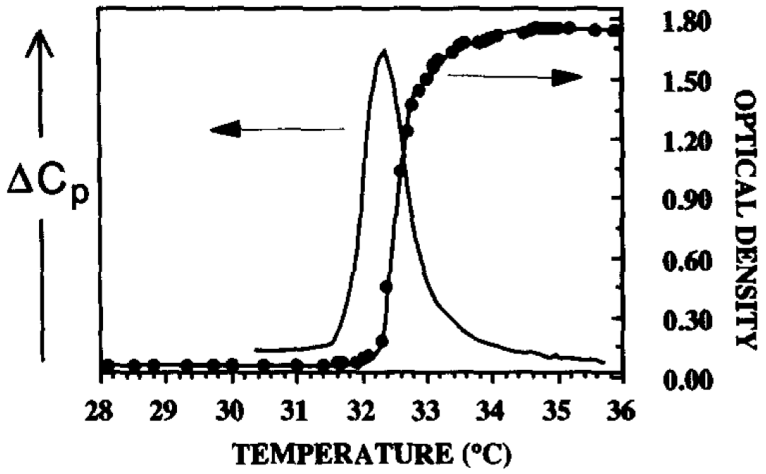


Figure 1.9: Cloud point and DSC measurements of pNIPAM, reprinted with permission from Elsevier [10].

Thermo-responsive gels have been applied in soft robotics (56–62), drug delivery (59, 63–68) and surgery [69]. Fig. 1.10 and Fig. 1.11 demonstrate two example applications. The LCST transition is accompanied by partial dehydration of the polymer network, which can be applied for the development of polymeric temperature sensors. The incorporation of dye molecules in the polymer side chain provides a temperature sensitive change in color

or fluorescence, as shown in Fig. 1.10 [11]. Zhang et al. [12] prepared micro-structured, temperature-sensitive pNIPAM hydrogels and utilized these porous hydrogels as matrices for controlled protein release (Fig. 1.11).

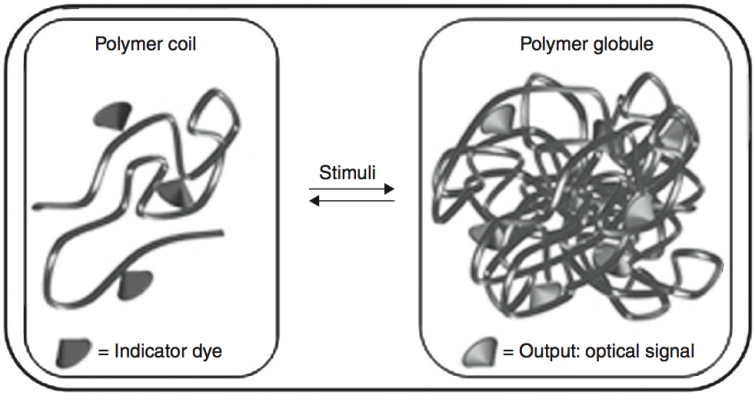


Figure 1.10: Schematic representation of the concept of polymeric sensors based on a polymer phase transition. The thermoresponsive structure and solvatochromic dyes provide a temperature sensitive change in color or fluorescence, figure adapted from [11].

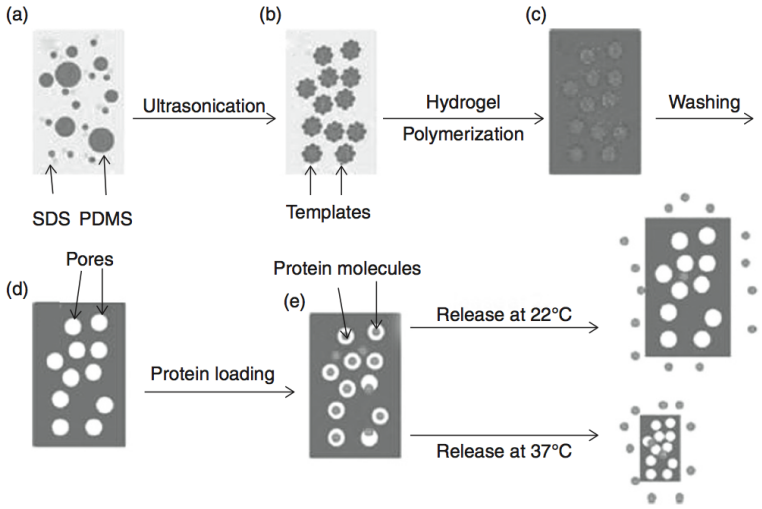


Figure 1.11: (a-e) Schematic illustration of preparation of porous smart pNIPAM hydrogels and their application for protein release, reprinted with permission from Elsevier [12].

1.4 Shape Memory Polymers

1.4.1 The Shape Memory Effects

Shape memory polymers can be easily programmed to a temporary shape at high temperature because of their low stiffness. Upon cooling to low temperature, the temporary shape is fixed and stabilized. When heated to high temperature again, the material recovers to its permanent configuration, as if it ”memorizes” this shape. This is known as the shape memory effect. The motion during recovery can be prescribed because it reverses the motion in programming. An example of a shape-memory tube is shown in Fig. 1.12 [13].

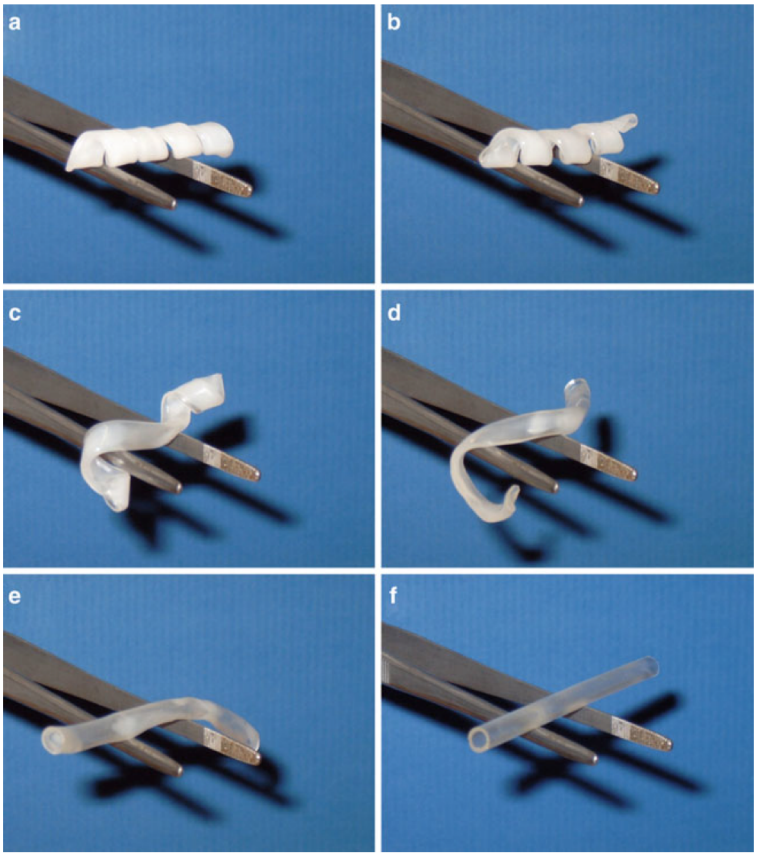


Figure 1.12: Recovery of a shape-memory tube: (a)-(f) images taken in the recovery process, reprinted with permission from Taylor & Francis [13].

Fig. 1.13 depicts the shape memory cycle. The polymer network involves netpoints determining the permanent configuration and chain segments that respond to external stimuli. Netpoints can be formed by covalent bonds (chemical crosslinks) for thermosets or physical entanglements for thermoplastics. Examples of polymer networks that exhibit the shape memory effect are shown in Fig. 1.14 [13, 70, 71].

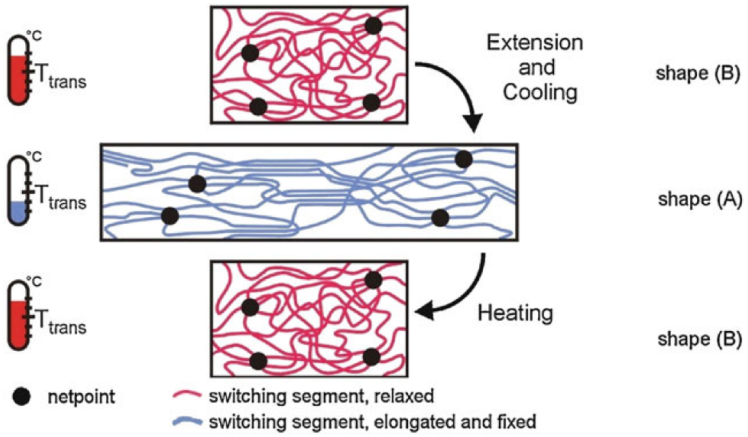


Figure 1.13: Polymer network undergoing shape memory effect, reprinted with permission from Taylor & Francis [13].

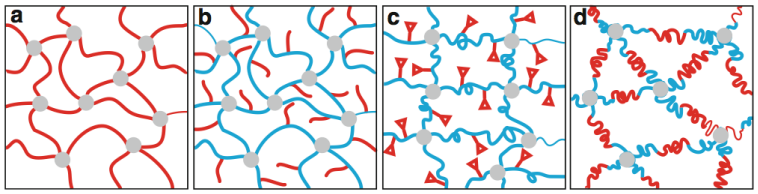


Figure 1.14: Examples for polymer network exhibiting shape memory effects (a) stimuli sensitive segments connecting netpoints, (b) side chains as stimuli sensitive segments, (c) functional groups as stimuli sensitive segments, (d) ABA triblock segments connecting netpoints, reprinted with permission from Taylor & Francis [13].

SMPs exhibit large deformation in response to various kinds of stimuli. Their structures can be conveniently tailored to meet requirements of a wide range of applications. Additional functional groups can be implemented in the polymer network to achieve multifunctionality [72]. Sample applications of SMPs include packaging foils [13] and reusable composite tooling [73]. Further examples include smart adhesives [74] and self-disassembling fasteners for mobile phones [75].

1.4.2 Glass Transition, Physical Aging and Mechanical Rejuvenation

1.4.2.1 Glass Transition

Shape memory polymers transform from rubbery state at high temperature to glassy state at low temperature, a phenomenon known as the glass transition. The corresponding transition temperature is denoted as T_g. The material undergoes a significant change in a number of properties. For example, the stiffness reduces dramatically when the material is heated above T_g. Figure 1.15 shows the storage modulus and $\tan\delta$ of polystyrene at different temperatures [14]. At low temperatures, the polymer chains have low mobility and the material exhibits a high glassy modulus. At elevated temperatures, the polymer structure obtains more mobility resulting in a dramatic decrease in the modulus. Another physical property that changes with the glass transition is the coefficient of thermal expansion (CTE). Above T_g, the polymer has a relatively large rubbery CTE, which can be determined from a linear fit of the specific volume versus temperature curve in the high temperature region. Below T_g, the CTE reduces to a smaller value that can be determined from a linear fit of the same curve in the low temperature region. The intersection of the two straight lines gives the glass transition temperature, the value of which depends on the cooling rate, as shown in Fig. 1.16 [14]. A lower cooling rate results in a lower value of T_g.

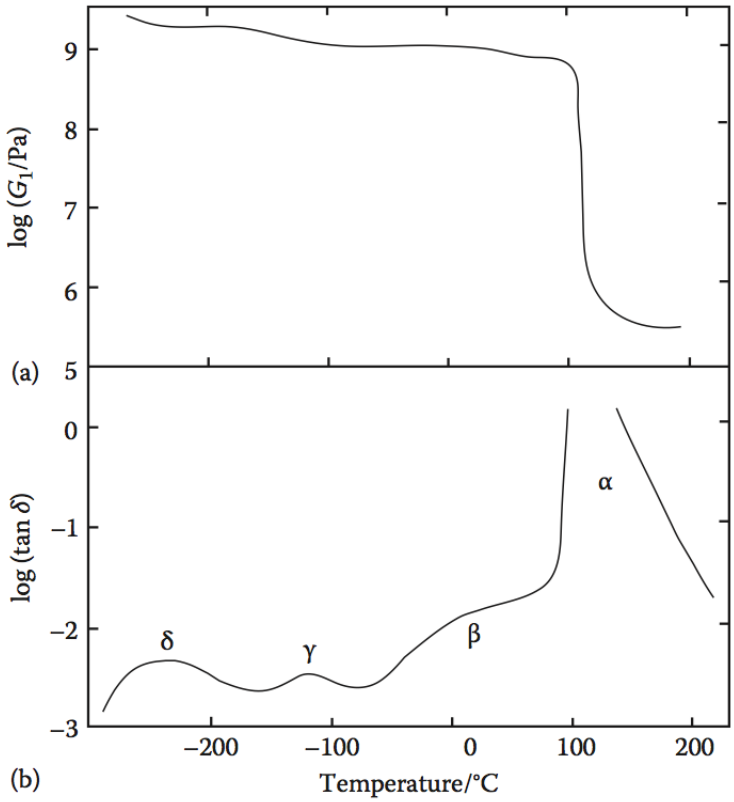


Figure 1.15: Change of (a) storage modulus and (b) $\tan\delta$ at differenrt temperature for polystyrene, reprinted with permission from Taylor & Francis [14].

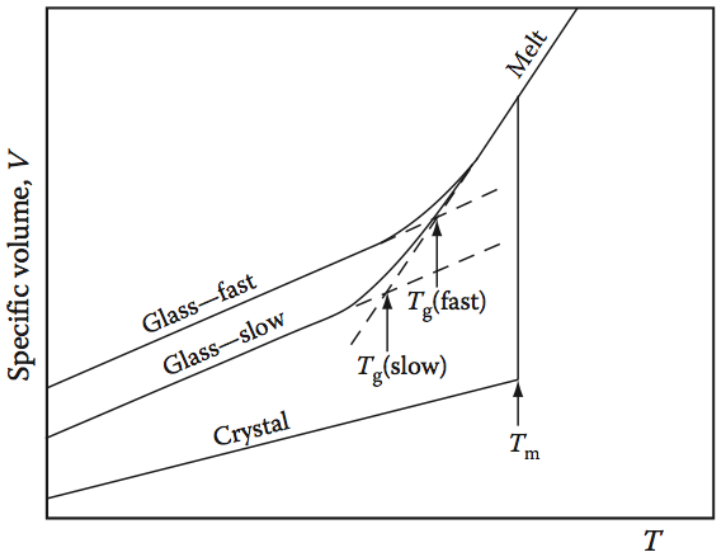


Figure 1.16: Specific volume vs. temperature for different materials, reprinted with permission from Taylor & Francis [14].

From a thermodynamic point of view, transitions can be categorized as first order transitions and second order transitions. The Gibbs free energy G is a function of independent state variables, volume V, pressure P and temperature T. For first order transitions, G is a continuous function of V, P and T, but the first partial derivatives of G, including the volume $V = \left(\frac{\partial G}{\partial P}\right)_T$, the enthalpy $H = \left(\frac{\partial (G/T)}{\partial (1/T)}\right)_P$ and the entropy $S = -\left(\frac{\partial G}{\partial T}\right)_P$, are discontinuous. For second order transitions, the Gibbs free energy and its first order derivatives are continuous, but the second order derivatives, including the thermal expansion coefficient $\alpha = \frac{1}{V}\left(\frac{\partial}{\partial T}\left(\frac{\partial G}{\partial P}\right)_T\right)_P$, the compressibility $\kappa = \left(\frac{\partial^2 G}{\partial P^2}\right)_T$ and heat capacity $C_p = -T\left(\frac{\partial^2 G}{\partial T^2}\right)_P$, become discontinuous. Fig. 1.17 shows the temperature dependence of the above thermodynamics quantities through the glass transitions [14]. As shown in the figure, the glass transition shares some features of a second-order transition. However,

Fig. 1.16 shows that T_g depends on cooling rate, which is different from a standard second order transition [14].

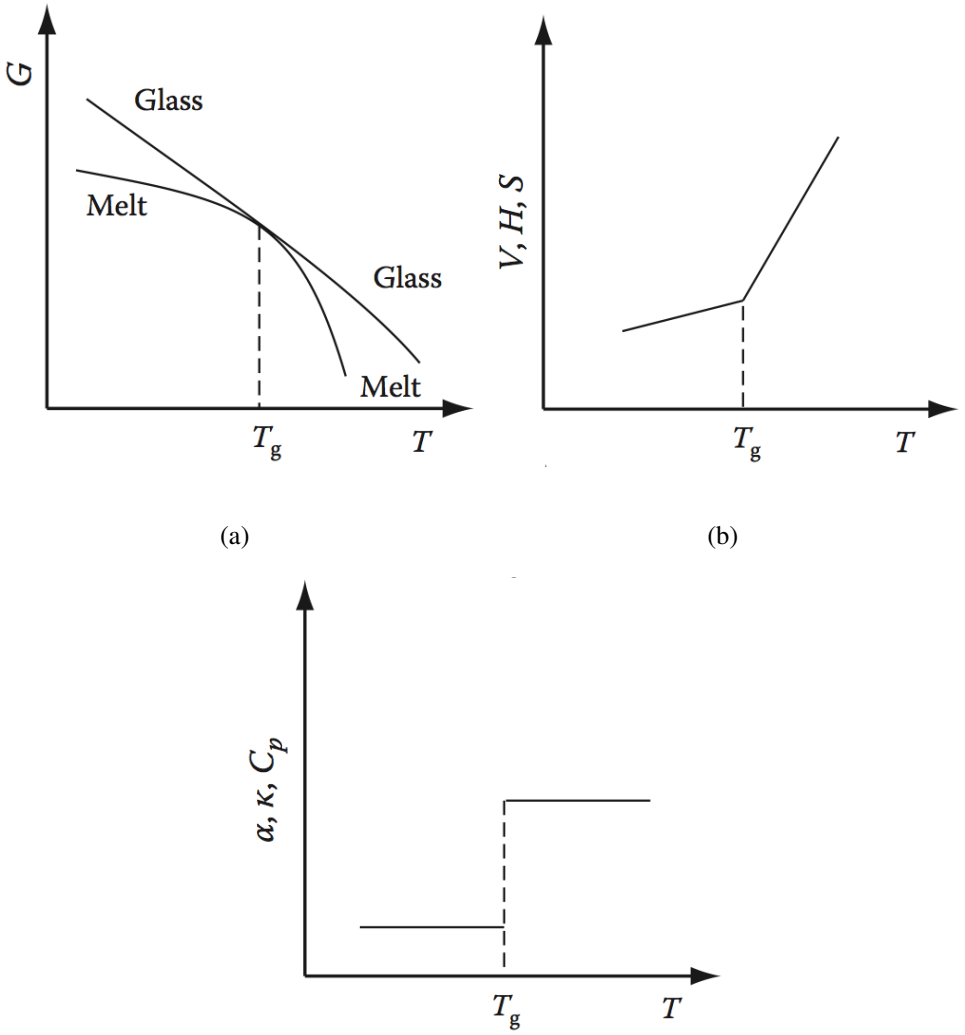


Figure 1.17: Schematic illustration of the temperature dependence of thermodynamic parameters during the glass transition, reprinted with permission from Taylor & Francis [14], (a) Gibbs free energy, (b) First order derivatives of Gibbs free energy (V, H, S), (c) Second order derivatives of Gibbs free energy (α, κ, C_p).

1.4.2.2 Physical Aging and Mechanical Rejuvenation

When cooled below T_g, the polymer becomes trapped in a nonequilibrium state because of the reduced chain mobility. The nonequilibrium polymer structure gradually evolves to equilibrium, a process known as structural relaxation or physical aging. The process is accompanied by changes in macroscopic properties such as stiffness and enthalpy (76–80). As shown in Fig. 1.18 [15], aged specimens exhibit higher yield stresses because of a larger resistance to mechanical deformation. In differential scanning calorimetry (DSC) measurements, aged specimens exhibit an endothermic overshoot near T_g, as shown in Fig. 1.19(a) [16], as the result of the reduction in specific enthalpy due to structural relaxation.

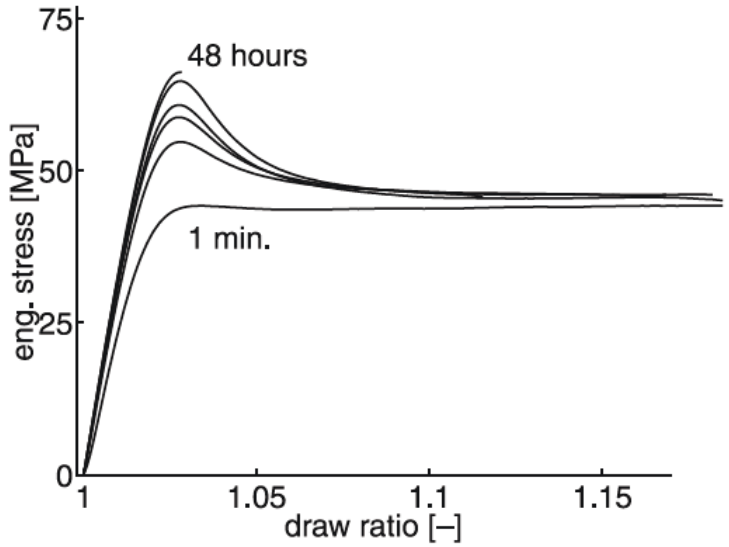
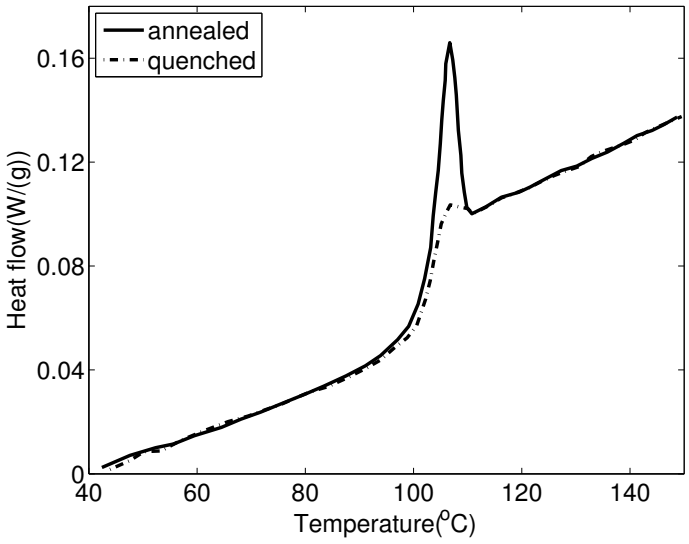


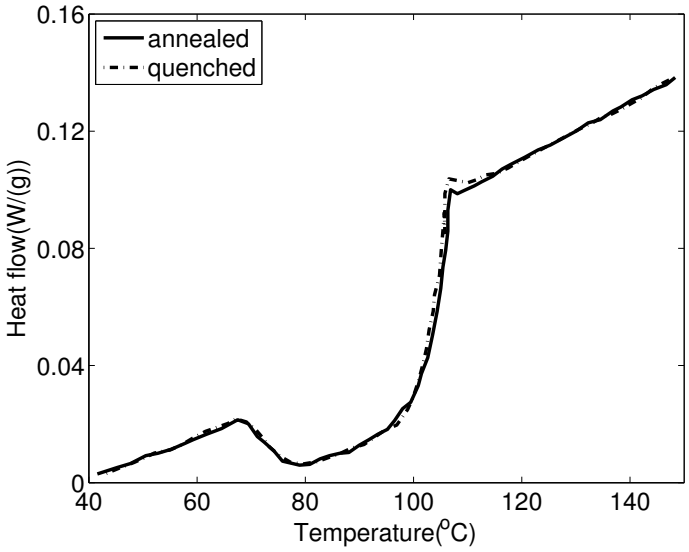
Figure 1.18: Uniaxial tension stress-strain curves of polystyrene specimens at different aging time, reprinted with permission from Elsevier [15].

Chain mobility of a polymer network recovers when heated above T_g, a process known

as thermal rejuvenation, or mechanically deformed below T_g, a process known as mechanical rejuvenation. Mechanical rejuvenation leads to strain-softening and erases the previous thermal history [16, 81]. As shown in Fig. 1.18, the flow stress of specimens with different aging time collapses to the same curve after mechanical deformation. Fig. 1.19(b) shows the DSC curves of quenched and annealed polystyrene specimens after 25% uniaxial compression [16]. The results for both quenched and annealed specimens are nearly the same because of the effects of mechanical rejuvenation.



(a)



(b)

Figure 1.19: DSC scans of annealed and quenched polystyrene with a heating rate of 10 °C/min, data extracted from [16], (a) Without deformation (b) After 25% compression.

1.5 Objectives of this Work

The aim of this work is to develop constitutive models for various kinds of soft active materials that are capable of accurately describing the materials' physical-mechanical behavior. To achieve this goal, we explored and developed a set of methods that can be applied to different kinds of soft materials. The developed methodology covers different stages of the study, from material characterization to constitutive modeling, from simulation to experiments.

In Chapter 2, we investigated the temperature dependent viscoelastic behavior of dielectric elastomers and the effects of viscoelasticity on the electro-actuation behavior. We measured the viscoelastic relaxation spectrum of VHB 4905 and applied the results to a discrete multi-process viscoelastic model. The model generally showed good quantitative agreement with experimental measurements in terms of both pure-mechanical and electro-mechanical behaviors of the material. The model was able to qualitatively capture the dependence of the electric breakdown time on voltage and pre-stretch.

In Chapter 3, we studied thermoresponsive hydrogel bilayer plates with soft and stiff segments that exhibit bidirectional and biaxial curving. We investigated the mechanism behind the deformation mode through finite element simulation. We concluded that the direction of bending was determined by the distribution of the swelling ratio through the thickness of the composite plate and can be altered by imparting a swelling gradient through the thickness. Biaxial bending occurred when the segments were arranged such that the long axis of the patch was perpendicular to the long axis of the plate. The curvaure of the

bent plate was controlled by the segment spacing, and a large segment spacing led to the development of wrinkles during bending.

In Chapter 4, we extended the effective temperature theory for the inelastic behavior through the glass transition in [82] to large strain and developed a constitutive model that is capable of describing strain hardening behavior in a thermodynamically consistent manner. The model incorporates two mechanisms: one represents the stretching and orientation of the polymer network, which leads to the development of a backstress; the other one represents the network relaxation, which accounts for the temperature and rate dependence of strain hardening. The model was applied to simulate the thermomechanical behavior of polycarbonate. All of the model parameters were determined through standard thermomechanical tests. The simulation results showed good agreements with experiments and the temperature dependence, strain rate dependence and strain state dependence of hardening were quantitatively captured.

Chapter 5 presents conclusions about the common methods that were applied in Chapter 2, 3 and 4. It also summarizes the key results and discussed the limitations of this work. The chapter ends with a discussion of future research directions.

Appendix A provides constitutive equations that incorporate thermal deformation in the model developed in Chapter 4. It also includes the method and results of determining the coefficient of thermal expansion. Appendix B summarizes the algorithms of the finite element implementation for the constitutive model in Chapter 4, including the procedure of solving the internal variables and the construction of the tangent modulus.

Chapter 2

The Temperature Dependent Viscoelastic Behavior of Dielectric Elastomers

In this chapter, we investigated the temperature dependent viscoelastic behavior of dielectric elastomers and the effects of viscoelasticity on the electro-actuation behavior. We performed dynamic thermo-mechanical analysis to measure the master curve of the stress relaxation function and the temperature dependence of the relaxation time of VHB 4905, a commonly used dielectric elastomer. The master curve was applied to calculate the viscoelastic spectrum for a discrete multi-process finite deformation viscoelastic model. In addition, we performed uniaxial creep and stress relaxation experiments and electrical actuation experiments under different prestretch conditions. The measured spectrum was ap-

plied to predict the experimental results. Generally, the model produced good quantitative agreement with both the viscoelastic and electro-actuation experiments, which shows the necessity of using a multi-process relaxation model to accurately capture the viscoelastic response for VHB. However, the model under-predicted the electro-actuated creep strain for high voltages near the pull-in instability. We attributed the discrepancies to the complex boundary conditions that were not taken into account in the simulation. We also investigated the failure of VHB membrane caused by viscoelastic creep when prestretched and subjected to constant voltage loading. The experimental time to failure for the specimens decreased exponentially with voltage, which agreed well with the predictions of the model.

This chapter is reprinted from [83], copyright (2015), with permission from ASME.

2.1 Introduction

Dielectric elastomers (DE) can deform when exposed to an applied electric field. The materials are capable of large deformation and have highly attractive features such as fast response, light weight, low cost and good electromechanical conversion efficiency [32,84]. When prestretched and sandwiched between two compliant electrodes, dielectric elastomer membranes have demonstrated voltage induced areal expansions up to 158% [85]. These features make dielectric elastomers attractive for soft robotic actuators [2,47,86], artificial limbs [87, 88], energy harvesters [89, 90], adaptive optics [91–93], Braille displays [94], and bio-stimulation pads [95,96].

CHAPTER 2. THE TEMPERATURE DEPENDENT VISCOELASTIC BEHAVIOR OF DIELECTRIC ELASTOMERS

Applying an electric field across the thickness of a dielectric elastomer generates a Maxwell stress that reduces the thickness of the film and causes the area to expand. As the elastomer thins, applying the same voltage leads to an increasing higher electric field. This positive feedback between deformation and electric field can lead to failure by pull-in instability and dielectric breakdown. When the film becomes unstable, complex 3D wrinkling patterns appear [97]. It has been revealed that prestretches can help to eliminate the pull-in instability [98] and improve electric breakdown strength [8, 99].

The elastic behavior of dielectric elastomers has been studied extensively [100–103], and the viscoelastic behavior of this material has gained increased scrutiny in recent works [104, 105]. Experiments have shown that viscoelasticity can significantly affect the electromechanical behavior. It has been observed in experiments that viscoelastic creep under a constant voltage actuation can induce the pull-in instability and electric breakdown [106]. Failure caused by pull-in instability and dielectric strength is a major limitation in the application of dielectric elastomers. Therefore accurate characterization of the viscoelastic behavior is important for the development of dielectric devices. Zhang and Chen proposed a viscoelastic model for the dielectric elastomer balloon using a free energy method. [107] Zhao et al. developed a nonequilibrium thermodynamic theory for the viscoelastic behavior of dielectric elastomers [108]. Park and Nguyen developed a dynamic finite element method to investigate the effect of viscoelasticity on the development of instabilities and electromechanical actuation [109, 110]. Lochmatter et al. [111] developed a visco-hyperelastic model to study a DE strip actuator under sinusoidal excitation. Yang

et al. [112] developed a nonlinear finite deformation viscoelastic model of dielectric membranes using Christensen's theory of viscoelasticity. Wissler and Mazza [113] used quasi-linear viscoelastic constitutive models with different energy formulations to predict creep behavior of VHB under different voltage levels. Recently, Kollosche et al. [114] applied a single process viscoelastic model to study the wrinkle-to-wrinkle transition of dielectric elastomers.

Viscoelastic models of dielectric elastomers typically apply a single relaxation process to describe the creep, stress relaxation, and rate-dependent stress response over a narrow range of time scales [108, 114]. A few models have applied multiple relaxation processes [109, 113, 115], up to 4, to describe a wider range of behavior and time scales, such as $10^{-1} - 10^{3}$ seconds. The parameters for the relaxation times and moduli were fit to mechanical creep or stress relaxation data [115] or voltage-induced electrical actuation data [113]. These approaches are limited in that they do not describe completely the broad distribution of relaxation processes nor the temperature dependence of the viscoelastic behavior. Michel et al. measured the elastic modulus of VHB 4910 by uniaxial tension experiments at different temperatures and observed a strong temperature dependence [116].

The aim of this work was to characterize the temperature-dependent viscoelastic spectrum of VHB, a widely used commercial dielectric elastomer. We applied the principles of time-temperature superposition (TTS) to construct the master curve of the relaxation modulus from relaxation tests at different temperatures. The master curve was applied to calculate the viscoelastic spectrum and temperature-dependence of the relaxation times for a

discrete multi-process viscoelastic model. We performed standard viscoelastic experiments (creep and stress relaxation) as well as voltage actuation experiments on VHB membranes. The measured spectrum was applied to simulate these experiments as well as experiments performed by [97, 106] to validate the model. Finally, we introduced an approach to selectively reduce and truncate the viscoelastic spectrum for a given time scale and temperature for computational efficiency, and demonstrated the shifting of the spectrum to describe the viscoelastic behavior at different temperatures.

2.2 Methods

2.2.1 Experimental Method

Commercially available VHB 4905 membranes (3M, USA) were used in all experiments. Membranes were cut to different sizes, as described below, according to needs of experiments.

2.2.1.1 Uniaxial Creep and Stress Relaxation Experiments:

The creep and stress relaxation response of VHB was measured using a dynamic mechanical analyzer (DMA TA Q800, Newcastle, DE 19720, US). Film specimens, 15 mm long, 6.0 mm wide, and 0.5 mm thick were cut from the VHB 4905 tape and mounted in the film tension grips of the DMA leaving an 8.0 mm gauge length.

The stress relaxation response was measured at different temperatures to construct the master curve of the relaxation modulus. The temperature was decreased from 60 ^{o}C to -40 ^{o}C in steps of 5 ^{o}C. Each temperature step was held for 15 minutes to reach thermal equilibrium. The specimens were stretched at a strain rate of 1%/s up to a strain of 0.01 and held at this strain while the stress was measured for 20 minutes. The relaxation modulus was calculated as the uniaxial engineering stress divided by the applied strain and was plotted as function of time. Values of the relaxation modulus measured at different temperatures were horizontally shifted to the reference temperature 20 ^{o}C, according to the principles of the time-temperature superposition, to form a master curve for the time-dependent relaxation modulus. Specifically, we defined a function $\alpha(T)$ for the temperature-dependent shift factor. To determine the shift factor at the temperature T, the relaxation modulus measured at T was shifted along the horizontal log-time axis by the logarithm of the shift factor, $\log \alpha(T)$, to join the curves for the relaxation modulus generated by data from previous temperatures. The logarithmic shift factor was $\log \alpha(T) < 0$ for temperatures lower than the reference temperature and $\log \alpha(T) > 0$ for temperatures higher than the reference temperature. Shift factors were adjusted until all data points formed a continuous, smooth master curve at the reference temperature.

We performed additional stress relaxation experiments at large strains and creep experiments for model validation. For the stress relaxation experiments, the film specimens were equilibrated at 20 ^{o}C, stretched to 100% strain at a rate of 12.5%/s and relaxed for 2 hours. For the creep experiments, film specimens were equilibrated at 20 ^{o}C and subjected to a

uniaxial force, which linearly increased from zero to 0.17 N with a loading rate of 0.01 N/s and then held for 6 hours. Uniaxial stretch during creep was recorded as a function of time.

2.2.1.2 Electrical Actuation Experiments:

Square specimens ranging from 63.5 mm to 127 mm in size were cut from the VHB tape. The side length was decided according to the prestretch level that would be applied in a particular experiment. A cross was drawn in the center of the film and four points were drawn around the cross and used as optical markers to measure the prestretch applied in the experiments. The specimen was uniaxially or equibiaxially stretched to the desired prestretch and attached to a rigid frame which is a square with side length of 190.5 mm. VHB is an adhesive tape, that provided a strong attachment to the frame. A circular electrode made from carbon conductive grease (MGchemicals, 846-1P) with a diameter of 10 mm was applied to the center of the prestretched film on both sides. The area of the central electrode was relatively small compared to the dimensions of the prestretched specimen. The prestretched film was allowed to relax 1.5 hours to reach an equilibrium stress state. A voltage was applied across the electrodes using a high voltage power supply (Gamma High Voltage Research, ES50P-5W, Ormond Beach, FL 32174, US). The experiments investigated applied voltages from 2.5 kV to 6 kV. A digital camera mounted above specimens was used to image the deforming specimens every 8 s to 20 s. The images were analyzed using GIMP2 [1] to measure the major and minor axes of the deformed electrode (Fig. 2.1).

[1] www.gimp.org

The principal stretches were calculated from the ratio of deformed major and minor axes to the undeformed diameter of the electrode.

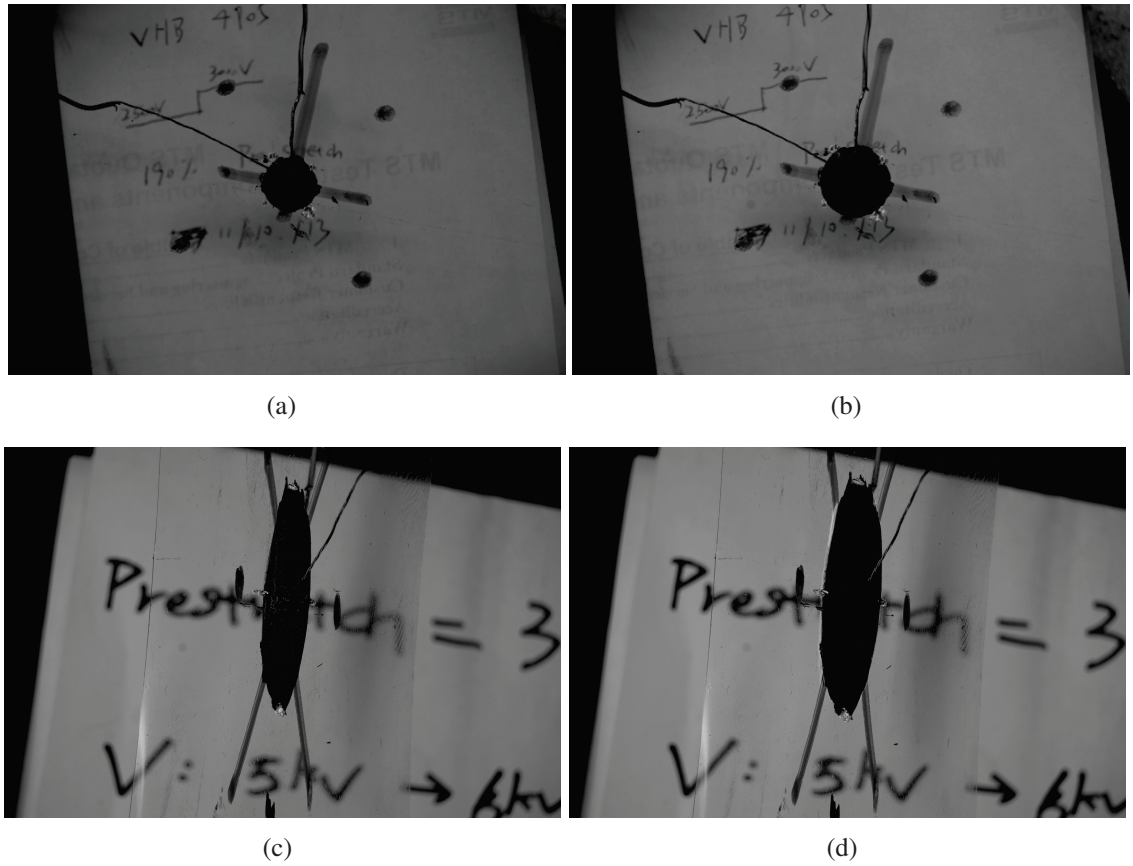


Figure 2.1: Images taken from actuation experiments: (a)(b) equibiaxial prestretch, before and after actuation (c)(d) uniaxial prestretch, before and after actuation.

2.2.2 Constitutive Model

We applied the analytical model developed by Suo [7] to describe the viscoelastic biaxial stress response of an incompressible dielectric elastomer membrane subjected to an equilibrium prestretch and a sudden applied voltage change (Fig. 2.2). The biaxial stress

state was characterized by in-plane stretches λ_1 and λ_2, and an out-of-plane stretch λ_3. For a membrane with applied prestrech λ_1^{pre} and λ_2^{pre}, we can define relative stretches as $\lambda_1^{rel} = \lambda_1/\lambda_1^{\text{pre}}, \lambda_2^{rel} = \lambda_2/\lambda_2^{\text{pre}}$ which represent stretches purely from electric actuation.

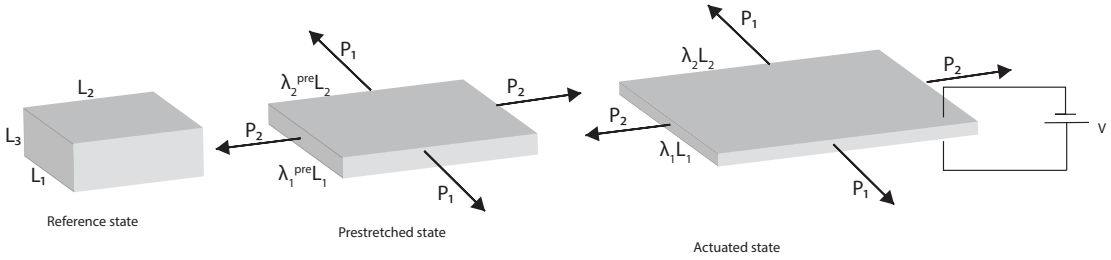


Figure 2.2: Model of electromechanical couple of DE membrane.

In the model, we assumed that VHB 4905 is an incompressible and isotropic material. Thus, $\lambda_3 = \lambda_1^{-1}\lambda_2^{-1}$. The permittivity of the material was assumed to be $\varepsilon = 3.98 \times 10^{-11}$ F/m [8], which gave a dielectric constant of $\varepsilon_d = 4.5$. We also assumed that the permittivity is independent of deformation for simplicity. Previous studies have assumed both a linear and nonlinear dependence on deformation [117] [118]. Moreover, we assumed that VHB is an ideal dielectric elastomer, such that the total free energy density of dielectric elastomers can be decomposed into mechanical and electrical parts:

$$W = W_{mech} + W_e \tag{2.1}$$

The electrical contribution to the free energy density is described by a quadratic potential,

$$W_e = \varepsilon E^2/2 \tag{2.2}$$

where E is electrical field. The electric displacement is given as $D = \varepsilon E$.

The electric field energy density of an ideal dielectric elastomer can be written in terms of the nominal electric field $\tilde{E}$ and electric displacement $\tilde{D}$ as:

$$W_e = \frac{\tilde{D}^2}{2\varepsilon}\lambda_1^{-2}\lambda_2^{-2} = \frac{\varepsilon\tilde{E}^2}{2}\lambda_1^2\lambda_2^2. \tag{2.3}$$

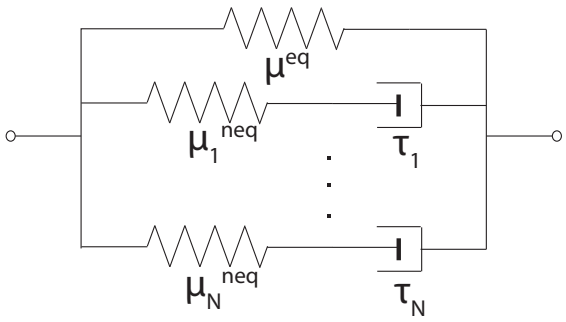


Figure 2.3: Standard rheological model

The mechanical behavior of the dielectric elastomer is described by the multi-process viscoelastic model illustrated in Fig. 2.3. The model consists of a parallel distribution of Maxwell models acting in parallel with an equilibrium spring. The equilibrium spring is characterized by an equilibrium shear modulus of μ^{eq} while the Maxwell models are described by a shear modulus of μ_k^{neq} and relaxation time of τ_k. The temperature dependance of the relaxation time was described as $\tau_k(T) = \tau_k^{\text{ref}}\alpha(T)$, where τ_k^{ref} is the relaxation time at the reference temperature (20 $^{\text{o}}$C) and $\alpha(T)$ is the shift factor measured in Sec. 2.2.1.1.

We assumed a multiplicative decomposition of the deformation gradient into elastic λ_{1e}^k and viscous components ξ_2^k, such that $\lambda_{1e}^k = \lambda_1/\xi_1^k, \lambda_{2e}^k = \lambda_2/\xi_2^k$. The N relaxation processes represent a discrete approximation of a continuous distribution of relaxation times. Using a larger number N of processes provide a more accurate description of the time-dependent relaxation behavior at the expense of computational time. In Sec. 2.2.3, we present a method to determine the parameters of the relaxation spectrum from the master curve of the relaxation modulus. We also investigated in Sec. 2.3.4 the effect of coarsening and altering the range of the discrete relaxation spectrum on the simulation results of the viscoelastic behavior of VHB.

We assumed that the mechanical strain energy density can be additively split into equilibrium and nonequilibrium components, $W_{mech}(\lambda_1,\lambda_2,\xi_1^k,\xi_2^k) = W_{mech}^{\text{eq}}(\lambda_1,\lambda_2) + \sum W_{mech}^{\text{neq}}(\lambda_{1e}^k,\lambda_{2e}^k)$. We adopted the Gent model for both the equilibrium and nonequilibrium components to describe the stiffening of the material when approaching the limiting stretch [119]:

$$\begin{aligned} W_{mech} = & -\frac{\mu^{\text{eq}} J_{\text{lim}}^{\text{eq}}}{2}\log\left(1-\frac{\lambda_1^2+\lambda_2^2+\lambda_1^{-2}\lambda_2^{-2}-3}{J_{\text{lim}}^{\text{eq}}}\right) \\ & -\sum_{k=1}^{N}\frac{\mu_k^{\text{neq}} J_{\text{lim}}^{\text{neq}}}{2}\log\left(1-\frac{\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3}{J_{\text{lim}}^{\text{neq}}}\right), \end{aligned} \tag{2.4}$$

where $J_{\text{lim}}^{\text{eq}}$ and $J_{\text{lim}}^{\text{neq}}$ are the limiting stretches of the equilibrium spring and non-equilibrium components, ξ_1^k and ξ_2^k denote kth viscous stretches in two in plane principle directions. We have assumed that the nonequilibrium processes share the same limiting stretch $J_{\text{lim}}^{\text{neq}}$ for simplicity. This model is an extension of the viscoelastic models developed by Zhao et

al. [108] and Park and Nguyen [109] that used a single exponential relaxation process to describe the viscoelastic behavior. Substituting eq.(2.3) and (2.4) into eq.(3.1), gives the total free energy density function. The principal first Piola-Kirchhoff engineering stresses were defined from the free-energy density, as [7]:

$$\begin{aligned} s_1 &= \frac{\partial W(\lambda_1,\lambda_2,\xi_1^k,\xi_2^k,\tilde{D})}{\partial \lambda_1} \\ s_2 &= \frac{\partial W(\lambda_1,\lambda_2,\xi_1^k,\xi_2^k,\tilde{D})}{\partial \lambda_2}, \end{aligned} \tag{2.5}$$

where $s_1 = \frac{P_1}{L_2 L_3} = \frac{\sigma_1}{\lambda_1}$ and $s_2 = \frac{P_2}{L_1 L_3} = \frac{\sigma_2}{\lambda_2}$ are engineering stresses in two principle directions. Evaluating eq. (2.5) and applying the relation between the engineering and Cauchy stresses gives:

$$\begin{aligned} \sigma_1 + \varepsilon E^2 &= \frac{\mu^{\text{eq}}(\lambda_1^2 - \lambda_1^{-2}\lambda_2^{-2})}{1-(\lambda_1^2+\lambda_2^2+\lambda_1^{-2}\lambda_2^{-2}-3)/J_{\text{lim}}^{\text{eq}}} + \sum_{k=1}^{N} \frac{\mu_k^{\text{neq}}[\lambda_1^2(\xi_1^k)^{-2} - \lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}} \\ \sigma_2 + \varepsilon E^2 &= \frac{\mu^{\text{eq}}(\lambda_2^2 - \lambda_1^{-2}\lambda_2^{-2})}{1-(\lambda_1^2+\lambda_2^2+\lambda_1^{-2}\lambda_2^{-2}-3)/J_{\text{lim}}^{\text{eq}}} + \sum_{k=1}^{N} \frac{\mu_k^{\text{neq}}[\lambda_2^2(\xi_2^k)^{-2} - \lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}}, \end{aligned} \tag{2.6}$$

Following [120], we apply the following evolution equation for the viscous stretches:

$$\eta_k \frac{d\xi_i^k}{\xi_i^k dt} = \frac{1}{2}\sigma_{i_k}^{\text{neq}}, \text{for } i = 1,2 \tag{2.7}$$

where $\eta_k(T) = \mu_k^{\text{neq}}\tau_k(T) = \mu_k^{\text{neq}}\tau_k^{\text{ref}}\alpha(T)$ is the shear viscosity, $\sigma_{i_k}^{\text{neq}}$ is the nonequilibrium deviatoric component of the Cauchy stress in equation (2.6) and can be written as:

$$\sigma_{1_k}^{\text{neq}} = \frac{1}{3}\left(\frac{2[\lambda_1^2(\xi_1^k)^{-2}-\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}} - \frac{[\lambda_2^2(\xi_2^k)^{-2}-\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}}\right)\mu_k^{\text{neq}}$$
$$\sigma_{2_k}^{\text{neq}} = \frac{1}{3}\left(\frac{-[\lambda_1^2(\xi_1^k)^{-2}-\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}} + \frac{2[\lambda_2^2(\xi_2^k)^{-2}-\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2]}{1-[\lambda_1^2(\xi_1^k)^{-2}+\lambda_2^2(\xi_2^k)^{-2}+\lambda_1^{-2}\lambda_2^{-2}(\xi_1^k\xi_2^k)^2-3]/J_{\text{lim}}^{\text{neq}}}\right)\mu_k^{\text{neq}}, \tag{2.8}$$

To determine the criteria for pull-in instability, we calculated the Hessian matrix of the free energy density function (3.1).

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 W}{\partial \lambda_1^2} & \frac{\partial^2 W}{\partial \lambda_1 \partial \lambda_2} & \frac{\partial^2 W}{\partial \lambda_1 \partial \widetilde{D}} \\ \frac{\partial^2 W}{\partial \lambda_1 \partial \lambda_2} & \frac{\partial^2 W}{\partial \lambda_2^2} & \frac{\partial^2 W}{\partial \lambda_2 \partial \widetilde{D}} \\ \frac{\partial^2 W}{\partial \lambda_1 \partial \widetilde{D}} & \frac{\partial^2 W}{\partial \lambda_2 \partial \widetilde{D}} & \frac{\partial^2 W}{\partial \widetilde{D}^2} \end{bmatrix}, \tag{2.9}$$

The components of $\mathbf{H}$ are functions of of $\lambda_1, \lambda_2, \xi_1^k, \xi_2^k, \tilde{D}$ and evolves with time. We used the following criteria for pull-in instability in our simulation. [108]:

$$\det \mathbf{H} = 0. \tag{2.10}$$

2.2.3 Parameter Determination

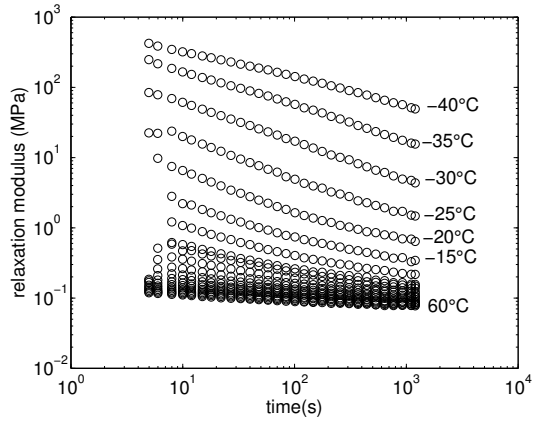
The master curve of the relaxation modulus was used to determine the equilibrium shear modulus and the parameters of the discrete relaxation spectrum $(\tau_k, \mu_k^{\text{neq}})$. We initially assumed a large number of relaxation processes, $N = 30$. Values of the relaxation modulus measured at different temperatures are plotted on a log-log scale in Fig. 2.4(a). The relaxation modulus was shifted horizontally to a reference temperature of 20 $^{\text{o}}$C to form the

master curve for the time-dependence of the relaxation modulus in (Fig. 2.4(b)). The material exhibited a broad glass relaxation spectrum, where the relaxation modulus spanned more than 13 decades of log time. The temperature dependance of the shift factor can be described by the WLF empirical relation,

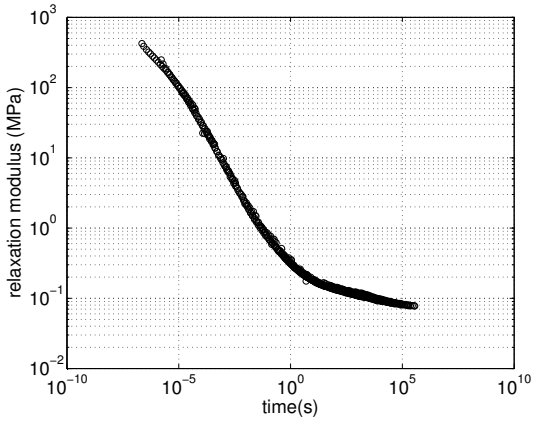
$$\log \alpha(T) = \frac{-C_1^{\text{ref}}(T - T^{\text{ref}})}{C_2^{\text{ref}} + T - T^{\text{ref}}}, \tag{2.11}$$

where C_1^{ref} and C_2^{ref} are the WLF constants for the reference temperature. Fitting the WLF relation to the shift factor data (Figure 2.5) gave $C_1^{\text{ref}} = 13.7$ and $C_2^{\text{ref}} = 187.1^{\text{o}}$C. The equilibrium modulus $E^{\text{eq}} = 0.078$ MPa was approximated from the plateau of the master curve at large times. Assuming incompressibility, the equilibrium shear modulus can be determined from the equilibrium Young's modulus as $\mu^{\text{eq}} = E^{\text{eq}}/3 = 0.026$ MPa. The nonequilibrium $E^{\text{neq}} = 689$ MPa was approximated by the maximum relaxation modulus of the master curve in Fig. 2.4(b) and applied to calculate the $\mu^{\text{neq}} = 230$ MPa assuming incompressibility. Physically, μ^{neq} represents the glassy modulus. However, we were unable to reach a plateau for the lowest temperature, -40 $^{\text{o}}$C, of the stress relaxation tests. At room temperature 20 $^{\text{o}}$C, the relaxation observed at -40 $^{\text{o}}$C corresponds to stress relaxation at 10^{-7} s, which is significantly smaller than the experimental time scale. Consequently, truncating the master curve at 10^{-7} s should not significantly affect the model predictions of material behavior and instability at room temperature.

We developed a 2nd order approximation method based on Schwarzl et al. [121] to determine the discrete relaxation spectrum [122]. A 7^{th} order polynomial was first fit



(a)



(b)

Figure 2.4: Relaxation modulus as a function of time, (a) measured for different temperature and (b) shifted to a reference temperature of 20^{o}C to form a master curve.

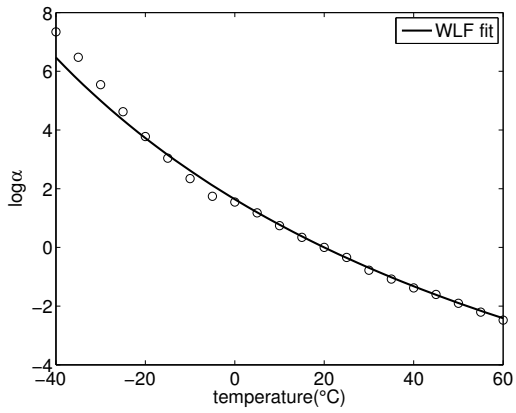


Figure 2.5: Shift factor as a function of temperature

to the master curve in Fig. 2.4(b). The resulting polynomial function was denoted as $\log G = f_0(\log t)$, where G is the relaxation modulus. The relationship between the relaxation modulus and a continuous relaxation spectrum was defined as [123]:

$$G(t) = \mu^{\text{eq}} + \int_0^\infty \frac{h(\tau)}{\tau^2} e^{-t/\tau} d\tau, \tag{2.12}$$

where $h(\tau)$ is the continuous relaxation spectrum. The cumulative relaxation spectrum was defined from $h(\tau)$ as,

$$H(\tau) = \int_0^\tau h(z) dz, \tag{2.13}$$

Evaluating $h(\tau)$ requires inverting the integral equation (2.12), which can be challenging. A number of approximations have been developed. In particular, we used a second order accurate approximation developed by Schwarzl and Staverman [121],

$$h(\tau) = -\frac{t}{2}G(t)[d\log G(t)/d\log t - (d\log G(t)/d\log t)^2 - (1/2.303)d^2\log G(t)/d(\log t)^2]|_{t=2\tau}, \tag{2.14}$$

Applying the polynomial fit to equations (2.14) and (2.13), we can calculate the continuous relaxation spectrum and cumulative distribution from polynomial f_0 as:

$$h(\tau) = -\frac{1}{2}10^{f_0(x)+x}\Big(f_0{}'(x) - (f_0{}'(x))^2 - (1/2.303)f_0{}''(x)\Big)|_{x=\log 2\tau}, \tag{2.15}$$

$$H(\tau) = 10^{f_0(x)}\Big(1 - f_0{}'(x)\Big)|_{x=\log 2\tau}, \tag{2.16}$$

The relaxation modulus of the discrete model can be written by assuming infinitesimal deformation as,

$$G_{disc}(t) = \mu^{\text{eq}} + \sum_k^N \mu_k^{\text{neq}} \exp(-t/\tau_k), \tag{2.17}$$

The discrete cumulative spectrum can be evaluated by combining equations (2.12) (2.13) and (2.17) as,

$$H_{disc}(\tau) = \sum_k^N \mu_k^{\text{neq}} \langle \tau - \tau_k \rangle - \mu^{\text{eq}}, \tag{2.18}$$

where $\langle \tau - \tau_k \rangle = 0$ for $\tau > \tau_k$. A power law distribution was applied for relaxation time [124]:

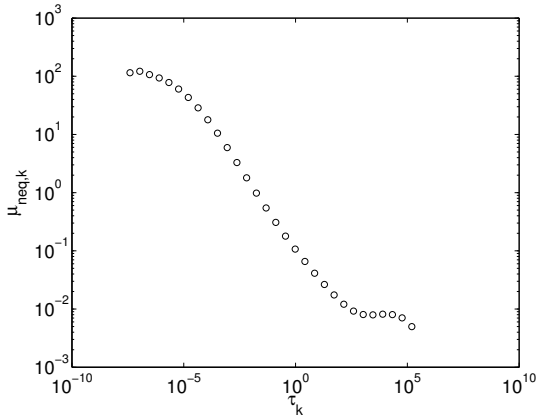
$$\tau_k = \tau_{\max}(\frac{\tau_{\min}}{\tau_{\max}})^{\frac{k-1}{N-1}}, \tag{2.19}$$

The upper and lower bounds of relaxation distribution, $\tau_{\max} = 1.59 \times 10^5$ s, $\tau_{\min} = 3.98 \times$

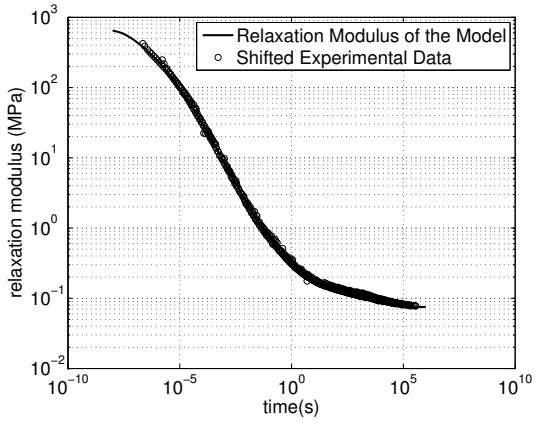
10^{-8} s, were determined according from the time range of master curve of the relaxation modulus (Fig. 2.4(b)). Finally, the nonequilibrium moduli μ_k^{neq} corresponding to the relaxation times τ_k were determined from the continuous cumulative distribution as follow: [124] [125]:

$$\begin{aligned}
\mu_1^{\text{neq}} &= \tfrac{1}{2}(H(\tau_1)+H(\tau_2)),\\
\mu_k^{\text{neq}} &= \tfrac{1}{2}(H(\tau_{k+1})-H(\tau_{k-1})), 1<k<N-1,\\
\mu_N^{\text{neq}} &= \mu^{\text{neq}} - \sum_{k}^{N-1} \mu_k^{\text{neq}}
\end{aligned} \tag{2.20}$$

The discrete cumulative spectrum forms a stepwise approximation of the continuous cumulative spectrum. Fig. 2.6(a) plots relaxation spectrum $(\tau_k, \mu_k^{\text{neq}})$ determined as described above. Note that μ_k^{neq} was maximum at the lower bound τ_{min} of the distribution of relaxation times, which indicates that relaxation spectrum extends to relaxation times lower than the chosen τ_{min}. However, relaxation times below the lower bound would relax too quickly to significantly affect the modeling results at room temperature. The spectrum was applied to equation (2.17) to evaluate the relaxation modulus of the discrete model. The results in Fig. 2.6(b) show excellent agreement with the measured master curve.



(a)



(b)

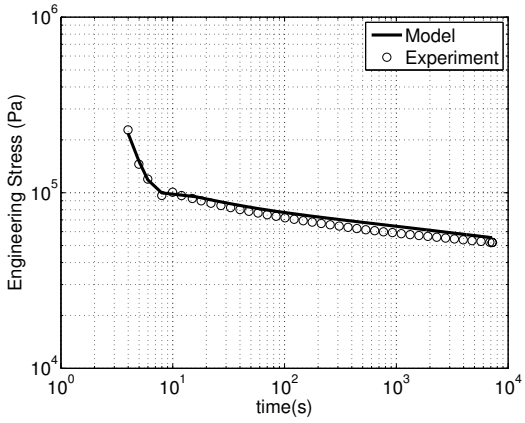
Figure 2.6: (a) Distribution of discrete viscoelastic spectrum $(\tau_k, \mu_k^{\text{neq}})$ (b) Comparison between master curves from experiments and the discrete model

2.3 Results and Discussion

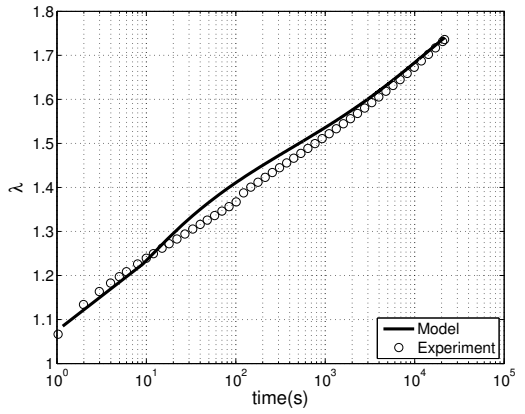
2.3.1 Creep and Stress relaxation

The viscoelastic model was applied using parameters determined in Sec. 2.2.3 to simulate the creep and stress relaxation experiments described in Sec. 2.2.1.1. In addition, we used $J_{\text{lim}}^{\text{eq}} = 110, J_{\text{lim}}^{\text{neq}} = 55$ in the simulations based on the previous work of Foo et al. [126]. It was verified through numerical tests that the simulations were insensitive to $J_{\text{lim}}^{\text{eq}}$ and $J_{\text{lim}}^{\text{neq}}$, which means changing these values has little effect on results presented in this chapter. Fig. 2.7(a) compares the results of the uniaxial stress relaxation experiment and simulation of VHB films stretched to 100% strain. The simulation used the same loading history prescribed in experiments. The stress response showed significant relaxation, decaying from 255 kPa to 52 kPa in 2 hours. The modeling results showed good agreement with the stress relaxation measurements.

Fig. 2.7(b) plots the creep response from experiments and modeling. The loading history of the experiment was applied for the simulations. The specimen exhibited significant creep throughout the experiment. The stretch increased nearly linearly in log time for the entire duration of the 6 hours of the experiment. The model predictions showed good agreement with the creep measurements.



(a)



(b)

Figure 2.7: (a) Relaxation of the uniaxial tension engineering stress response and (b) uniaxial tension creep stretch response, comparing experiments and model prediction.

2.3.2 Effect of Strain Rate

Next, we applied the model and parameters determined in Sec. 2.2.3 to simulate the uniaxial tension experiments of VHB 4905 at different strain rates by Plante and Dubowsky [97]. Figure 2.8 compares model prediction and experimental results for stretch rates 3.3×10^{-4}/s, 9.4×10^{-2}/s and 1.8/s. The model accurately predicted the stress-strain response of VHB for the two lower stretch rates up to a stretch of 4. The largest discrepancy between the experimental data and model prediction was 18% and occurred for the highest stretch rate which was 1.8/s.

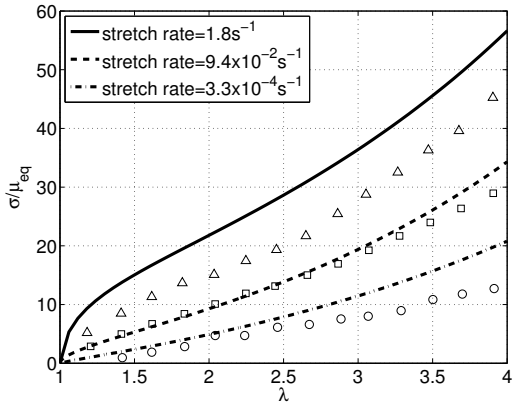


Figure 2.8: Normalized stress as a function of stretch from uniaxial tension with different stretch rates, comparing experiments and model prediction.

2.3.3 Voltage-induced Creep

Figure 2.9 plots the relative stretch as a function of time for the electro-actuation of an equibiaxially pre-stretched membrane, comparing experiments in Sec. 2.2.1.2 and model predictions. In the experiments, the specimen was prestretched equibiaxially to 1.9, relaxed

for 1.5 hours and then subjected to an applied voltage. The voltage was ramped from 0 to 2.5 kV in 10 seconds, held for 5 minutes, ramped to 3 kV in 10 seconds, and held until electric breakdown occurred. The loading history of the experiment was applied for the simulation. The electric field E was evaluated from the applied voltage as $E = V/l_3$, where l_3 is the deformed thickness of the film, then applied to eq. (2.6) to evaluate the equibiaxial stretch response. In addition, we simulated the electro-actuation experiments of [106], which subjected pre-stretched VHB membranes to the same pre-stretch, $V_{\text{low}} = 2.5$ kV and a higher $V_{\text{high}} = 3.5$ kV. Keplinger et al. [106] used a more accurate capacitive method to measure the stretch. The experiments and modeling results were compared in Fig. 2.9. The model predictions showed excellent agreement with experiments for the 2.5 kV step and also for the first 100 s of the 3.0 kV step. However, the model stretch prediction was significantly lower than experimental measurements near the onset of the pull-in instability observed by the accelerated creep rate.

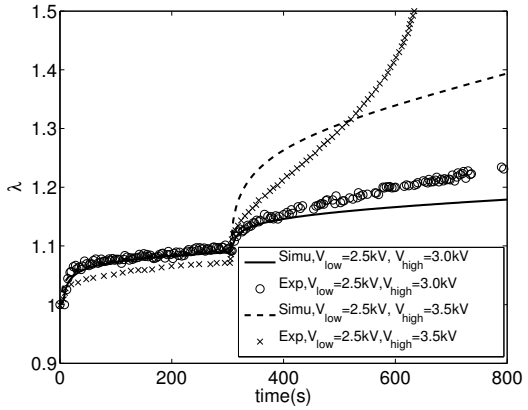


Figure 2.9: Relative stretch as a function of time ($\lambda_1^{\text{pre}} = \lambda_2^{\text{pre}} = 1.9$), comparing results from experiments and simulation.

2.3.4 Reduced Representations of the Stress Relaxation Spectrum

In above sections, we used a large number of relaxation processes, $N = 30$, to guarantee accurate description of the viscoelastic response. However, the larger number of relaxation processes resulted in long computational times. In this section, we investigated the effect of reducing the number of processes on the modeling results for voltage-induced creep. The majority of the distribution of relaxation times at 20 ^{o}C fell below 1 s, and should not significantly affect the long time electro-actuated creep response measured in the experiments. To demonstrate this, we determined the discrete spectrum for a reduced number N =20 of processes over the same span of relaxation times, $\tau_{\min} = 3.98 \times 10^{-8}$ s and $\tau_{\max} = 1.59 \times 10^{5}$ s. In addition, we evaluated the effects of using a truncated relaxation spectrum with a small number of processes, $N = 10, N = 5$ and $N = 3$. To obtain the truncated spectrum, we first determined the relaxation spectrum for $N_{total} = 13$ processes, then truncated this spectrum by picking the last 10 processes for $N = 10$, the last 5 processes for $N = 5$ and the last 3 processes for $N = 3$. Thus, the truncated distribution spans $\tau_{\min} = 3.16 \times 10^{-5}$ s, $\tau_{\max} = 1.59 \times 10^{4}$ s for the $N = 10$ case, $\tau_{\min} = 2.16$ s, $\tau_{\max} = 1.59 \times 10^{4}$ s for the $N = 5$ case and $\tau_{\min} = 185$ s, $\tau_{\max} = 1.59 \times 10^{4}$ s for the $N = 3$ case. The different cases were summarized in Table 2.1 and the spectra were plotted in Fig. 2.10.

The electro-actuation stretch response was computed for the 5 cases in Table 2.1 at room temperature $T = 20^{o}$C. The loading history was the same as introduced in Sec.

	N	$\tau_{\mathbf{max}}(\mathbf{s})$	$\tau_{\mathbf{min}}(\mathbf{s})$	$\mu^{\mathbf{neq}}(\mathbf{MPa})$
Case 1	30 (full range)	1.59×10^5	3.98×10^{-8}	230
Case 2	20 (full range)	1.59×10^5	3.98×10^{-8}	230
Case 3	10 (truncated)	1.59×10^4	3.16×10^{-5}	39.7
Case 4	5 (truncated)	1.59×10^4	2.16	0.110
Case 5	3 (truncated)	1.59×10^4	185	0.0247

Table 2.1: Parameters of spectra with different number of processes

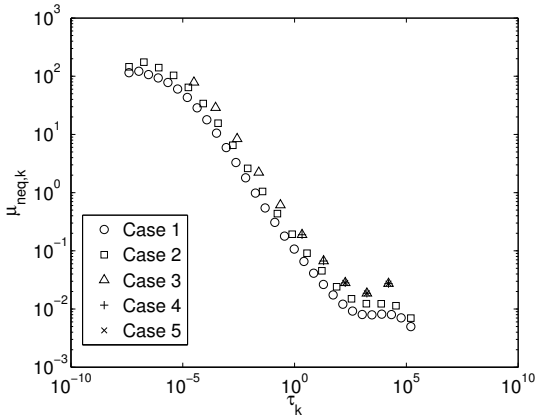


Figure 2.10: Distribution of discrete spectra for cases 1-5

2.3.3 with $V_{\text{low}} = 2.5$ kV and $V_{\text{high}} = 3.0$ kV. The results in Fig. 2.11 for case 1-4 showed negligible differences. The largest discrepancy among the first 4 cases was about 1%. Case 5 with only 3 nonequilibrium processes, however, cannot precisely describe the material's short time response because of the truncation. This shows the importance of using multiple nonequilibrium processes to accurately capture the viscoelastic response of VHB, which is in contrast to the single relaxation process models that have been used to-date [108, 114].

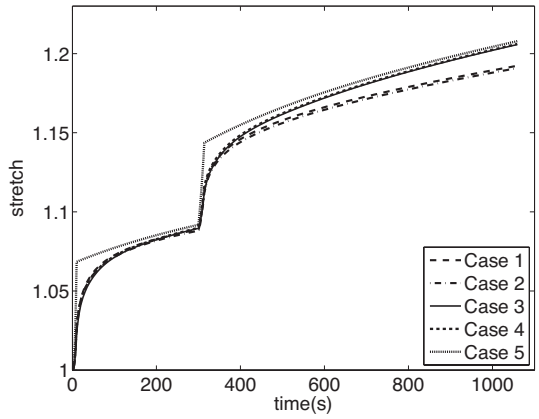


Figure 2.11: Voltage actuated creep simulation for cases 1-5 at 20$^{\text{o}}$C

The electro-actuation response was computed for a higher temperature, 60$^{\text{o}}$C comparing all five cases (Fig. 2.12). The spectra for all cases were shifted to 60$^{\text{o}}$C using the temperature dependent shift factors shown in Fig. 2.5. Larger differences were observed between the truncated and full relaxation spectrum. The largest difference between the full and truncated spectrum was 5%. At high temperature, the VHB material became more mobile, and accurate predictions were needed for the relaxation processes with smaller relaxation times.

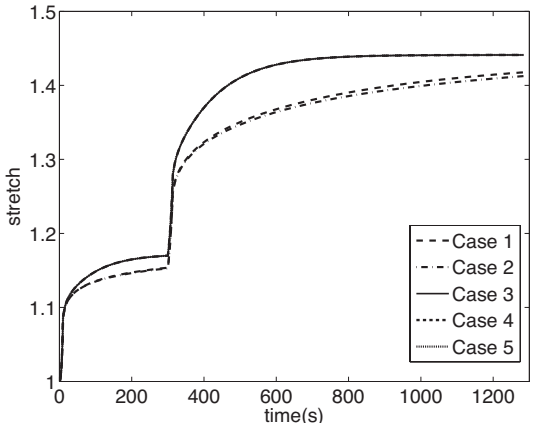
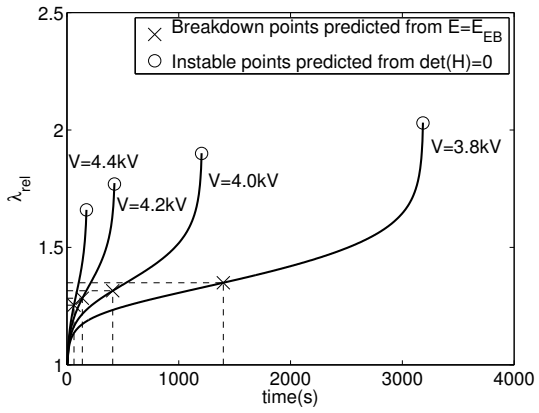


Figure 2.12: Voltage actuated creep simulation for cases 1-5 at 60^{o}C

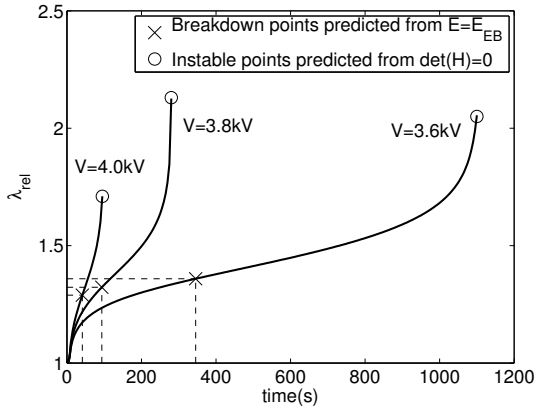
2.3.5 Voltage-induced Creep Failure

As shown in Sec. 2.3.3, the model was unable to capture the accelerated creep at the onset of the pull-in instability. We attributed this discrepancy to the complex boundary conditions of the experiments, which were neglected in the simulations. The simulations assumed homogeneous deformation and a uniform biaxial stress state in the film. In experiments, the voltage was applied to a small region of the film, and the resulting electroactuation caused the surrounding film to unload, which created a nonuniform deformation state.

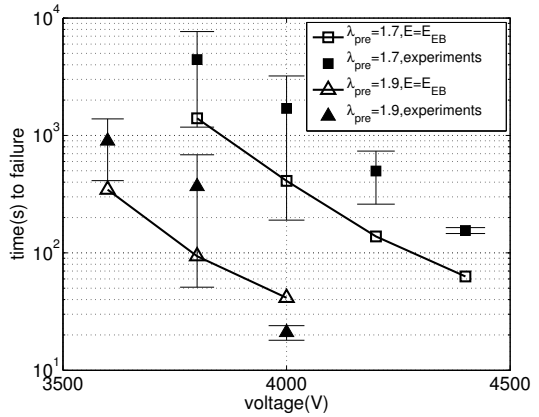
We applied the model to investigate the effects of prestretch and voltage on the failure of dielectric elastomers. The simulations considered two equibiaxial prestretch ratios, 1.7 and 1.9, and evaluated the time to creep-induced instability for different voltages. The criteria for pull-in instability was defined as $\det \mathbf{H} = 0$. In addition, we evaluated the time to electric breakdown for different voltages. The critical condition for electric breakdown



(a)



(b)



(c)

Figure 2.13: (a) Relative stretch from constant voltage actuation as a function of time, $\lambda^{\text{pre}} = 1.7, V = 3.8, 4.0, 4.2, 4.4$ kV, (b)relative stretch from constant voltage actuation as a function of time, $\lambda^{\text{pre}} = 1.9, V = 3.6, 3.8, 4.0$ kV, (c)time to failure as a function of voltage

was $\lambda = \lambda_{\text{EB}}$, where λ_{EB} is the breakdown stretch:

$$\lambda_{\text{EB}} = \sqrt{\frac{E_{\text{EB}} L_3}{V}}, \tag{2.21}$$

where L_3 represents the initial thickness and E_{EB} is the breakdown electric field. Kofod et al. [8] and Plante et al. [97] have measured E_{EB} as a function of prestretch. From their measurements, we used $E_{\text{EB}} = 40$ MV/m for a prestretch $\lambda^{\text{pre}} = 1.7$ and $E_{\text{EB}} = 48$ MV/m for $\lambda^{\text{pre}} = 1.9$. Fig. 2.13 (a)-(b) plotted the voltage-induced relative strain for $\lambda^{\text{pre}} = 1.7$ and $\lambda^{\text{pre}} = 1.9$. The time to break-down were marked using crosses on the curves in the figure, and the unstable limit points predicted from $\det \mathbf{H} = 0$ were marked using circles. In all cases breakdown happened before the pull-in instability which is coincident with our observations in the experiments that the stretch did not increase dramatically before electric breakdown occurred. Fig. 2.13(c) compared the time to failure measured from experiments and calculated from the electric breakdown criteria for the model. The experiments and modeling results showed the same trends. The time to electric breakdown decreased exponentially with voltage (linearly in a semi-log time). They also decreased with the prestretch.

2.4 Conclusions

We measured the viscoelastic relaxation spectrum of VHB 4905 and the temperature dependance of the relaxation times and applied the results to a discrete multi-process vis-

coelastic model. We showed that the viscoelastic spectrum can be truncated systematically to describe the time-dependent behavior in a more narrow time range, though the truncation was shown to require multiple, and not a single nonequilibrium process in order to accurately capture the viscoelastic response. Moreover, the spectrum can be shifted using the temperature-dependent shift factor to describe the time-dependent behavior at higher temperatures. The model generally showed good quantitative agreement with experimental measurements of electromechanical behaviors. The model was able to qualitatively capture the dependence of the electric breakdown time with voltage and pre-stretch.

Chapter 3

Bidirectional and Biaxial Curving of Thermoresponsive Bilayer Plates with Soft and Stiff Segments

Curved bilayer plates with soft and stiff segments are widely observed in nature, such as plant cell walls, insect exoskeletons and reptile skins. We report an unusual biaxial and bidirectional bending of microfabricated bilayer plates composed of a swellable, photopolymerized poly(N-isopropylacrylamide-co-acrylic acid) (pNIPAM-AAc) layer and a regular array of SU-8, a stiff, non-swellable epoxy. Hydrogels such as pNIPAM-AAc exhibit large and reversible swelling in an aqueous solvent in response to a temperature change through the lower critical solution temperature (LCST). The stimuli responsive behavior was harnessed in the composite structure containing materials with mismatched

swelling and elastic properties to produce actuation and mechanical motion. The structure undergoes reversible bending along two different axes in response to a temperature cycle through the LCST. Cooling the patterned bilayer structure leads to bending upwards about one axis, while heating leads to bending downwards about a different axis 90^o from the first. To understand the mechanism of this biaxial bending behavior, we developed a finite element model of the patterned bilayer structure. A constitutive model that combined the hyperelastic and swelling behavior was used to describe the thermoresponsive hydrogel. The model was applied to investigate the effects of geometric factors of patterned bilayer on the bending behavior of the composite structure.

This chapter is reprinted from [127], copyright (2017), with permission from Elsevier.

3.1 Introduction

Bilayer plate structures with soft and stiff segments can be widely found in nature at many different length scales. The composite design allows the structure to be both stiff for structural support and flexible for motion. Plant cell walls are composed of stiff cellulose fibrils embedded in a compliant and highly swellable matrix consisting of structural proteins, pectin and/or lignin [128–130]. The cell wall architecture results in anisotropic deformation of the plant tissue upon influx and efflux of water, enabling the release of ripe seeds from conifer cones [131] and dispersal of spore capsules from mosses [132,133]. The skin of reptiles [134, 135] and fish [136] are formed by overlapping stiff polygonal plates

composed of structural proteins, such as keratin, or mineralized tissues, such as dentin and enamel, that lie on top of a flexible dermal layer composed mainly of collagen and elastin. The scaly structure provides the animals with armor-like protection without compromising agile motion. Inspired by these natural materials, we designed a thermoresponsive thin film composed of stiff segments embedded in a soft hydrogel matrix that can be actuated through self-folding.

Self-folding of thin film structures allows for transformation from two dimensional (2D) to three dimensional (3D) geometries [137]. The thin-film structures that are typically micrometers to millimeters thick, are less defect-sensitive than molecular self-assembled structures because of the inherently larger domain sizes [138, 139]. One major advantage of thin-film structure, is that it can be manufactured using conventional 2D lithography, a fabrication process with low cost, high resolution and high throughput. Fernandes and Gracias [140, 141] provided a review of the design strategies for achieving self-folding of polymeric thin films. The basic strategy involves introducing a strain gradient through the thickness, through the deposition of material layers with mismatched elastic properties and actuation strain [142–144] or by introducing a crosslinking gradient through the thickness of a hydrogel film to produce a swelling gradient [3, 145]. Numerous studies have shown that complex 3D configurations and motions [146–152] can be achieved by patterning.

A significant limitation of hydrogel self-folding structures is their low stiffness. For example, microgrippers composed solely of soft hydrogels are too floppy to securely grip objects. The overall stiffness of the active structure can be enhanced by a composite design

that embeds stiff non-swellable polymer segments into the compliant hydrogel material [153–155]. The arrangement of the stiff segments strongly influences the folded shape. The presence of stiff line reinforcements in a thin hydrogel strip can produce rolling, twisting, or expansion during swelling depending the orientation of reinforcements. [156]

Inspired by natural armor systems, such as crocodile skin, we designed a thermoresponsive self-folding composite structure composed of stiff, non swellable SU-8 rectangular patches arranged regularly in a poly(N-isopropyla- crylamide-co-acrylic acid) (pNIPAM-AAc) matrix [157]. pNIPAM-AAc is a thermoresponsive hydrogel that undergoes a transition from a hydrophilic state to a hydrophobic state when the temperature increases above the lower critical solution temperature (LCST), resulting in a dramatic change in volume [10, 158–161]. While the composite design enhanced the overall stiffness of the plate, we found that the plate exhibited an unexpected bidirectional and biaxial bending behavior during temperature-induced swelling and deswelling. The goal of this chapter is to understand this unusual bending behavior and explore the design space of the composite structure using a combined experimental and computational modeling approach. We developed finite element models of the bilayer composite structure to simulate the self-folding behavior. A thermomechanical hyperelastic constitutive model was used to describe the equilibrium thermoresponsive swelling behavior of the pNIPAM-AAc hydrogel. We investigated the mechanism for biaxial bending by comparing the energy of the different equilibrium folded configurations. Similar energy minimization approaches have been applied in theoretical plate bending models [162–167] and finite element models [164, 165, 168–171]

to examine the effects of geometry and material properties on the equilibrium folding behavior of self-folding structures. We also applied the finite element models to investigate the effects of geometric factors, such as film aspect ratio, segment aspect ratio and segment spacing on the bending behavior to guide the design and application of the active composite structure.

3.2 Method

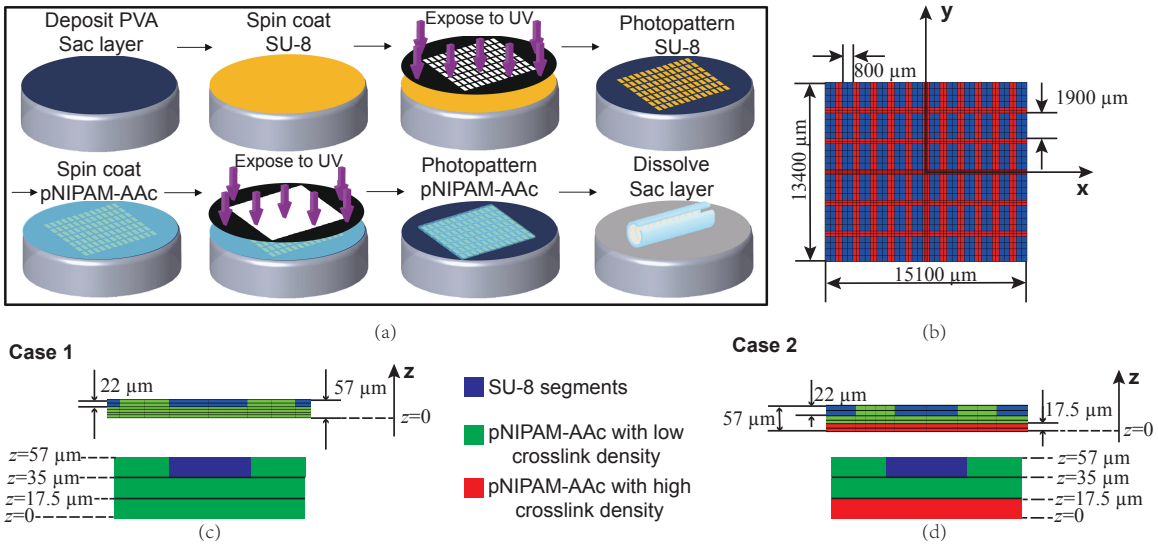


Figure 3.1: (a) Schematic of the microfabrication process for the soft-stiff hydrogel composite structure. (b) Finite element model of the composite structure, where red signifies the soft pNIPAM-AAc material and blue signifies the stiff SU-8 segments. Schematic of cross-section of the finite element model of the composite structure with (c) uniform pNIPAM-AAc properties and (d) with varying properties through the thickness.

3.2.1 Materials and Specimen Preparation

The patterned bilayer structure was fabricated using 2D lithography (Fig. 3.1 (a)). The structure consisted of a regular grid of stiff SU-8 segments, a non-swelling photoresist with a stiffness of 2 GPa, embedded in a poly (*N*-isopropylacrylamide-co-acrylic acid) (pNIPAM-AAc) layer. An 80% hydrolyzed polyvinyl alcohol (PVA, Sigma Aldrich, 9000 Da molecular weight) sacrificial layer was spin coated onto a silicon (Si) wafer at 2000 rpm and then baked at 115°C for 5 minutes on a hot plate. SU-8 2025 (MicroChem) photoresist was spin coated on the PVA coated Si wafer at 3000 rpm and then baked at 65°C for 2 minutes, 95°C for 5 minutes, and 65°C for 2 minutes. The SU-8 film was photopatterned using a mask aligner (Quintel) and a segmented rectangular shaped photomask and was exposed to 500 mJ/cm^2 ultraviolet (UV) light (365 nm) to initiate crosslinking. Moreover, post-baking was conducted at 65°C for 1 minute, 95°C for 5 minutes and, 65°C for 1 minute. Uncrosslinked regions of the SU-8 were completely dissolved using an SU-8 developer (MicroChem) for 1 minute and rinsed with acetone and isopropyl alcohol (IPA) in order, and dried with compressed air to obtain only the crosslinked SU-8 rectangular segment features on the PVA covered Si wafer. The wafer was treated with oxygen plasma (Plasma Etch) for 480 seconds at 30 W (RF power) before aliquoting pNIPAM-AAc solution on the wafer. The pNIPAM-AAc stock solution was prepared by following a previous protocol [60]. In brief, a mixture of 3 g NIPAM monomer (Scientific Polymer Products Inc.), 0.4 g pNIPAM (300k MW, Scientific Polymer Products Inc.), 0.18 g N, N-Methylenebis-Acrylamide (BIS-acrylamide, Aldrich), 7.5 ml organic solvent of n-butanol (Sigma), 0.31

ml acrylic acid (AAc, Aldrich) and 100 ml Irgacure 2100 (Ciba) were stirred overnight, and stored in room temperature until further use. In order to photopattern a thermally responsive pNIPAM-AAc layer on top of the SU-8 patterns, 1 ml of pNIPAM-AAc solution was cast onto the wafer and levelled for 1 minute. After aligning a second mask with the SU-8 patterns, the pNIPAM-AAc solution was exposed to 60 mJ/cm^2 UV light through a second dark field mask on top in non-contact lithographic mode using spacers. In this crosslinking process, the UV intensity decays along the thickness direction, which leads to a gradient of crosslink density. The uncrosslinked pNIPAM-AAc was washed off using IPA and then dried gently using compressed air. The photopatterned SU-8/pNIPAM-AAc bilayers were allowed to bond for 4 hours. The substrate was submerged in DI water to dissolve the PVA sacrificial layer. This yielded free standing bilayer plates in a swollen equilibrium state at room temperature. The samples were stored in DI water for up to three days at room temperature before the actuation experiments.

For the actuation experiments, SU-8/pNIPAM-AAc bilayer plates were immersed in DI water in a glass Petri dish on a hot plate. The water was heated from room temperature to 40°C. Finally, the hot plate was removed to allow the system to cool down to room temperature at ambient conditions. The deformation of the plates in the heating-cooling cycle was captured by a multi-zoom microscope (NIKON AZ 100).

3.2.2 Finite element model

The finite element geometry was developed using the dimensions of the pattern used to fabricate the composite structure (Fig. 3.1 (b)). The model consisted of a rectangular thin plate divided into two material domains, the pNIPAM-AAc hydrogel and the SU-8 rectangular segments, which were embedded in the hydrogel up to a depth of 22 μm. The mesh was discretized using trilinear hexahedral elements with mesh sizes shown in Fig. 3.1 (b). The boundary conditions were set as $u_x(0,0,0) = u_y(0,0,0) = u_z(0,0,0) = 0$. We considered two cases. In case 1, the pNIPAM-AAc was assumed to be uniform throughout the plate (Fig. 3.1 (c)). In case 2, we examined the effects of a gradient in swelling and elastic modulus through the thickness caused by the gradient in the crosslink density. The gradient was approximated by dividing the pNIPAM-AAc into two layers (Fig. 3.1 (d)). The bottom layer, ($0 < z < 17.5\mu$m), had higher crosslink density and thus was stiffer and swelled to a lesser extent than the upper layer.

We used a previously developed constitutive model [3, 155] to describe the thermally activated swelling behavior of hydrogels. We assumed that the free energy of the hydrogel system can be decomposed into a mechanical component for the stretching of the polymer chains and a chemical component for the mixing of the polymer and solvent,

$$\psi = \psi_e(\mathbf{F}, \varphi) + \psi_m(\varphi) \tag{3.1}$$

where $\mathbf{F}$ is the deformation gradient and φ is the polymer volume fraction of the polymer-

solute system. A quasi-incompressible model was used for the mechanical component of the free energy density,

$$\psi_e(\mathbf{F},\varphi) = \frac{G}{2}\left\{\lambda_1^2+\lambda_2^2+\lambda_3^2-3-2\ln(\lambda_1\lambda_2\lambda_3)\right\} + \frac{\kappa}{4}\left\{(\varphi\lambda_1\lambda_2\lambda_3)^2-2\ln(\varphi\lambda_1\lambda_2\lambda_3)-1\right\} \quad (3.2)$$

where G and κ are shear and bulk moduli, and $\lambda_1,\lambda_2,\lambda_3$ are the principle stretches. The free energy density of mixing was described using the Flory-Huggins theory [172],

$$\psi_m(\varphi) = \frac{RT}{\nu\varphi}\left\{(1-\varphi)\ln(1-\varphi)+\chi\varphi(1-\varphi)\right\}, \quad (3.3)$$

where ν is the volume of a solute molecule and $\chi(T)$ is the temperature-dependent Flory-Huggins interaction parameter [173],

$$\chi = \frac{1}{2}(\chi_L+\chi_H)+\frac{1}{2}(\chi_H-\chi_L)\tanh(\frac{T-T_{tran}}{\Delta T}). \quad (3.4)$$

The χ_L and χ_H are the Flory-Huggins interaction parameters at low and high temperature, T_{tran} represents the transition temperature and ΔT is the width of the transition region. The principle Cauchy stresses σ_a and chemical potential μ can be evaluated from the free energy density as,

$$\sigma_a = \frac{1}{\lambda_1\lambda_2\lambda_3}\lambda_a\frac{\partial\psi}{\partial\lambda_a}, a=1,2,3, \quad (3.5)$$

$$\mu = \frac{\partial \psi}{\partial c}, \tag{3.6}$$

where c represents the number of solute molecules per reference volume and is related to φ through $\varphi = 1/(1+\nu c)$. The constitutive model was implemented in TAHOE (Sandia National Laboratories) for finite element simulation.

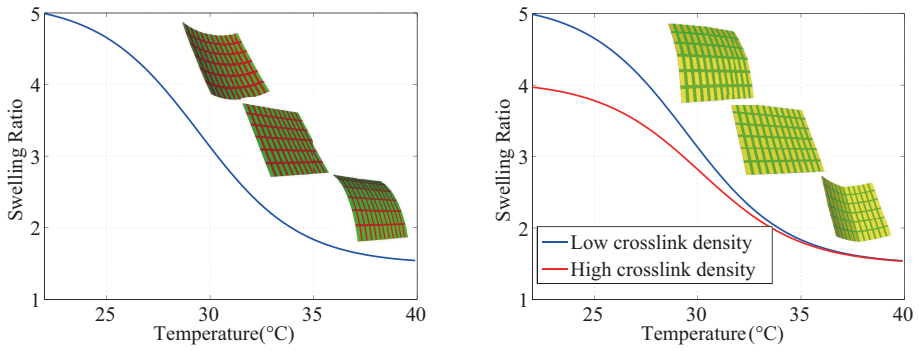
The simulation starts from a temperature of $T_0 = 29^{\circ}C$ at which the plate is assumed to be in a flat stress-free state. The temperature was decreased to $T_{low} = 22^{\circ}C$ or increased to $T_{high} = 40^{\circ}C$ from T_0 to produce swelling and deswelling of the hydrogel, respectively, thus introducing self-folding.

3.3 Results and Discussions

3.3.1 Simulation Results

The model parameters used to simulate case 1, with the uniform low crosslinked pNIPAM-AAc layer, and case 2, with the two layer pNIPAM-AAc structure, are summarized in Table 3.1 [3].

The volumetric swelling ratio for free swelling is plotted as a function of temperature in Fig. 3.2(a) and Fig. 3.2(b) for case 1, with uniform properties, and case 2, with nonuniform properties in the pNIPAM-AAc, respectively along with the swollen and deswollen configurations. As shown in Fig. 3.2(a) for case 1, the plate bent along the long axis up-



(a) (b)

Figure 3.2: Swelling ratio plotted as a function of temperature (a) for case 1 with uniform properties in the pNIPAM-AAc material and (b) for case 2 with a through-thickness variation in the properties of the pNIPAM-AAc material. Also plotted are the deformed configuration at 3 different temperatures. In (a), the plate bent along the long axis up towards the positive z-direction when swollen and along the short axis down towards the negative z-direction when deswollen. The opposite occurred in (b), where the plate bent along the short axis down towards the negative z-direction when swollen and along the long axis up towards the positive z-direction when deswollen.

	G	χ_L	χ_H	T_{tran}	ΔT
low crosslink density	74 KPa	0.57	0.98	307 °K	5 °K
high crosslink density	125 KPa	0.595	0.98	307 °K	5 °K

Table 3.1: The shear modulus (G) and Flory-Huggins parameters for pNIPAM-AAc with low and high crosslink density obtained from Yoon et al. [3].

wards, towards the positive z-direction, when swollen and along the short axis downwards, towards the negative z-direction, when deswollen. In case 2, the system also exhibited bidirectional and biaxial bending, but in an opposite sequence than observed for case 1. In the swollen state the plate bent along the short axis, down towards negative z-direction, while in the deswollen state it bent along the long axis up, towards positive z-direction.

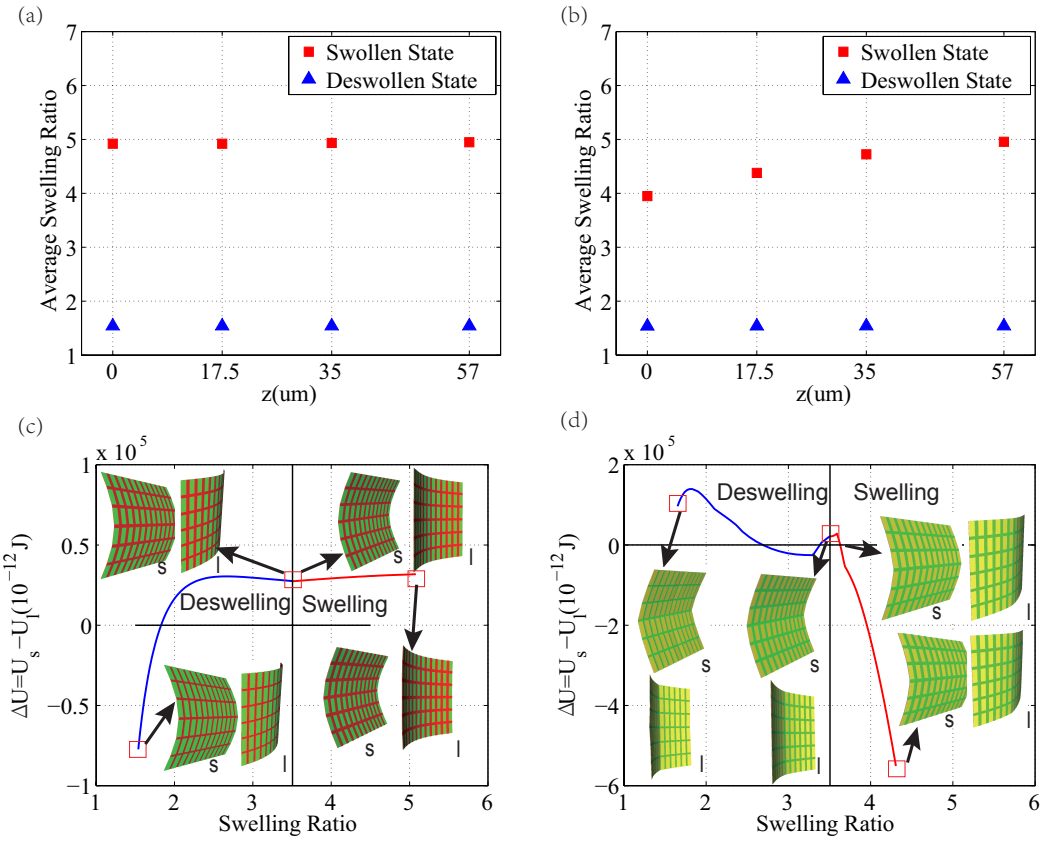


Figure 3.3: The area average swelling ratio at different positions along z for (a) case 1 with uniform properties in the pNIPAM-AAc material and (b) case 2 with a through-thickness variation in the properties of the pNIPAM-AAc material, where $z = 0$ corresponds to the bottom of the plate. The difference in the strain energy $\Delta U = U_s - U_l$ between bending along the short axis and along the long axis for (c) case 1 and (d) case 2. The dark gridline marks the 3.5 swelling ratio of the pNIPAM-AAc material at the initial temperature T_0.

To understand the sequence of bending directions, we examined the variation in the swelling ratio through the thickness for case 1 with uniform pNIPAM-AAc properties and case 2 with a variation in the pNIPAM-AAc properties along the thickness direction. The swelling ratio at different z-positions of the plate ($z = 0, 17.5, 35, 57\mu$m) were averaged over all the nodes at that z-position (Fig. 3.3(a, b)). In case 2, the top surface ($z = 57\mu$m) swelled more than the bottom ($z = 0$) because of the presence of the cross-linking gradient, which

explained the bending towards negative z in the swollen state. In case 1, the swelling ratio showed little variation through the thickness. As a result, the plate bent upwards towards the positive z-direction in the swollen state because the non-swelling SU-8 segments restricted the deformation of the top surface.

We also investigated the sequence of bending axes by applying finite element analysis to evaluate the difference in the strain energy for bending along the long axis compared to the short axis during swelling and deswelling (Fig. 3.3(c,d)). To force bending along either axis, we applied a cylindrical force field with a radius of 4266 μm to the initially flat bilayer plate at the initial temperature T_0. The force field was aligned with either the long or short axis of the bilayer plate depending on the bending axis of case 1 and 2. The force field was placed in contact with the plate then moved by a distance of 2133 μm, either in the positive or negative z-direction depending on the bending direction, to induce bending. The radius of the force field and its displacement were picked to reproduce the equilibrium curvature calculated in the finite element self-folding simulations at $T_{low} = 22^{\circ}C$ and $T_{high} = 40^{\circ}C$ for the swollen state and deswollen state respectively (Fig. 3.2). The mechanically bent plate was then subjected to a temperature change to allow the structure to swell or deswell under the constraint of the cylindrical force field. The strain energy density was integrated over the body to calculate the total strain energy U of the plate with subscript s referring to bending along the short axis and subscript l referring to bending along the long axis. The difference in U_s and U_l was plotted against the swelling ratio for case 1 in Fig. 3.3(c). A $\Delta U = U_s - U_l > 0$, signified that bending along the long axis was more energetically

favorable, while $\Delta U < 0$ indicated preferential bending along the short axis. For case 1, $\Delta U > 0$ during mechanical bending by the cylindrical force field, signifying that bending along the long axis resulted in a lower strain energy than bending along the short axis. From this starting point, the temperature was increased for deswelling or decreased for swelling. The ΔU was negative during deswelling and positive during swelling, which explained the sequence of bending axis shown in Fig. 3.2(a). Fig. 3.3(d) shows ΔU as a function of the swelling ratio of the top layer with the lower crosslink density for case 2. The opposite result was found for case 2. The ΔU was positive in the deswollen state and negative in the swollen state, which explained the bending sequence shown in Fig. 3.2(b).

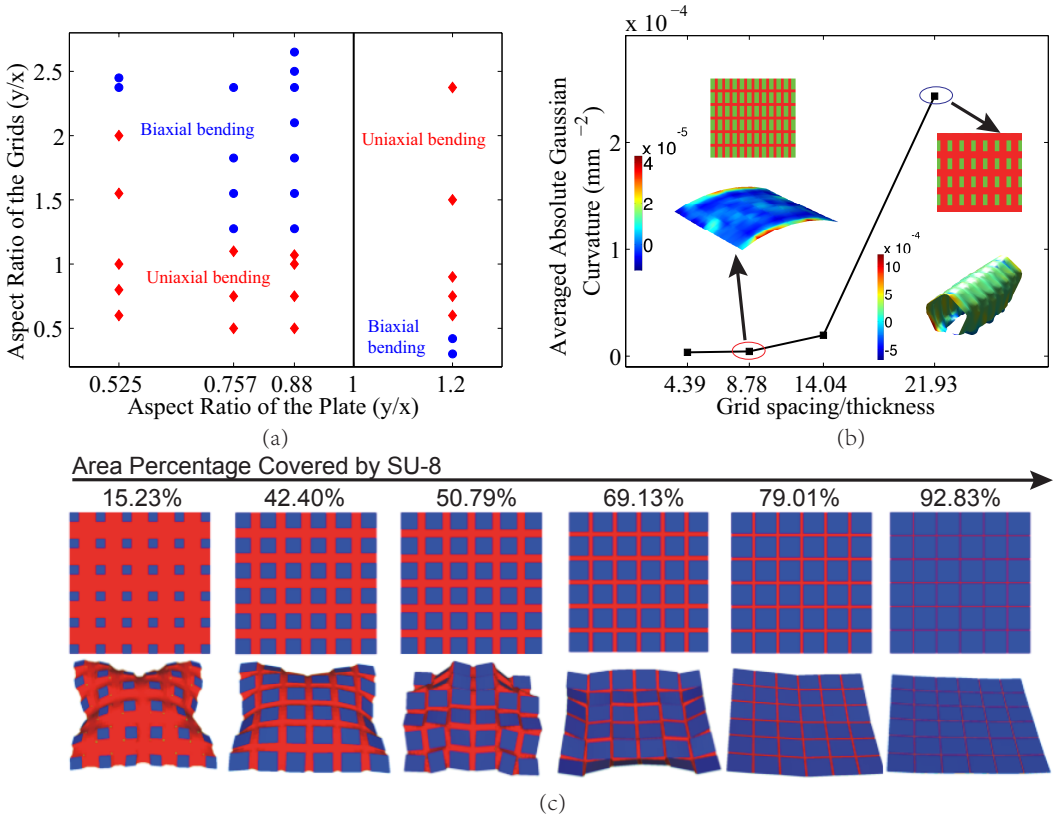
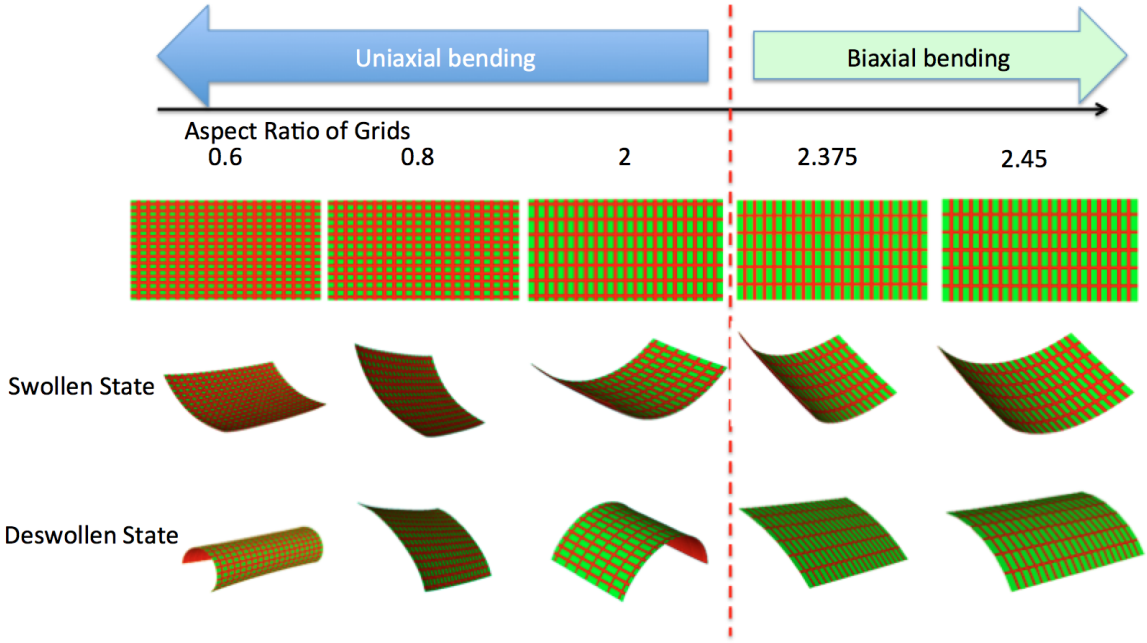
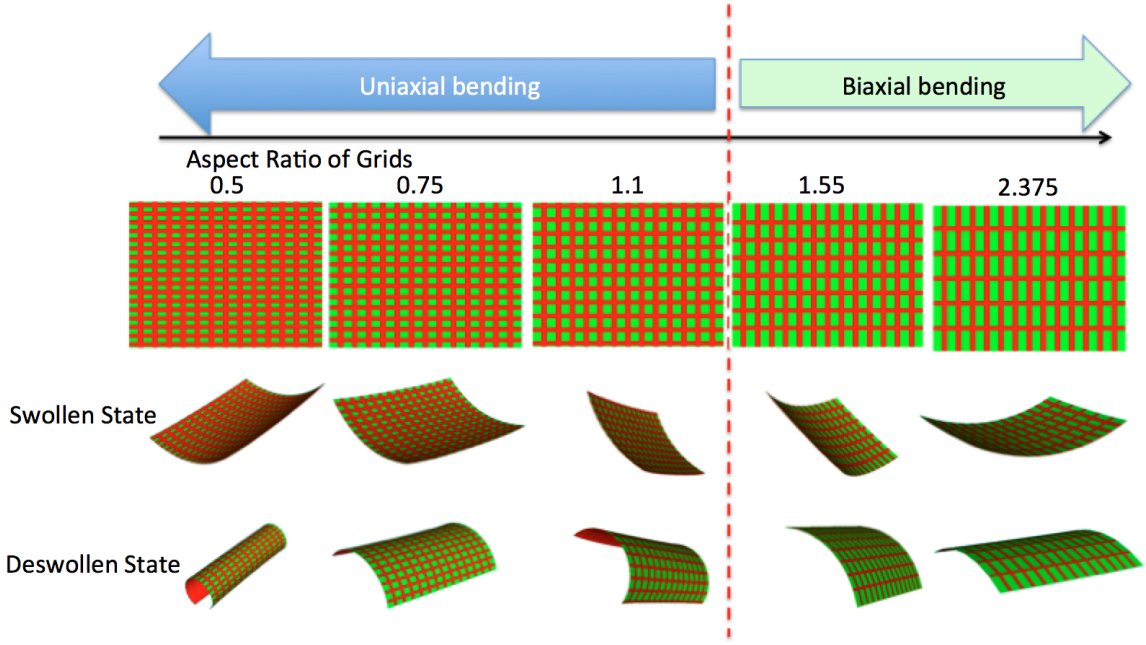


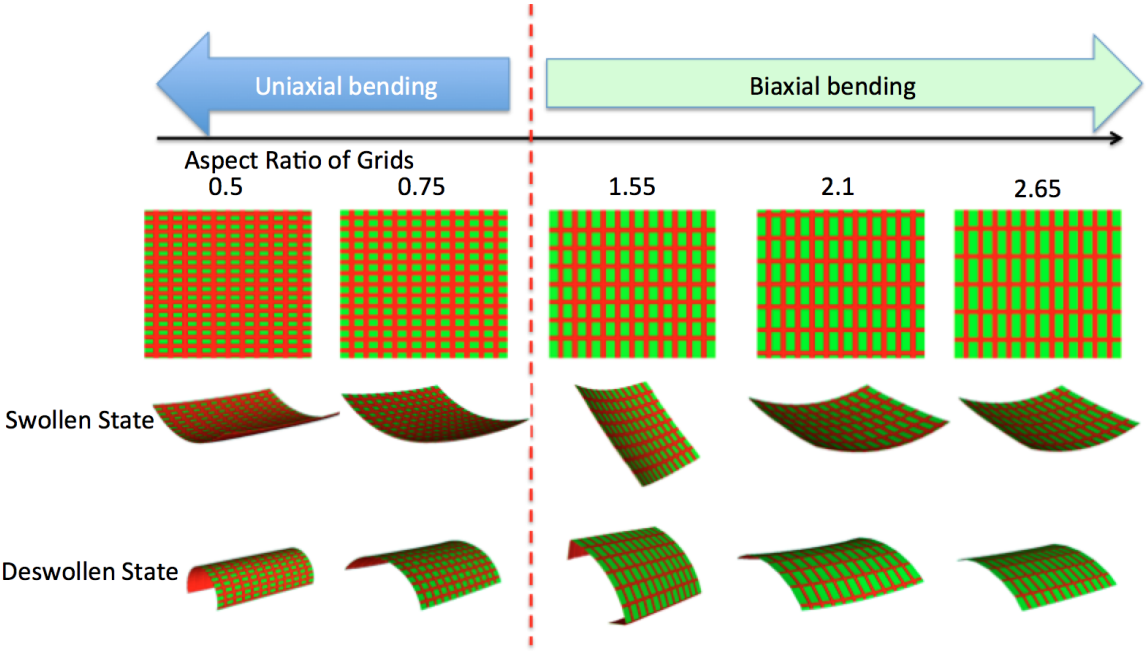
Figure 3.4: (a) A phase diagram for biaxial and uniaxial bending of the composite structure with different segment and plate aspect ratios. (b) The averaged absolute gaussian curvature, obtained by taking the average on the absolute value of the gaussian curvature over all the surface nodes, was plotted against the segment spacing normalized by the plate thickness. Contour plots of the gaussian curvature of the deswollen configurations are included for the normalized segment spacing of 8.78 and 21.93. (c) A schematic of the deswollen configurations of the square plates with square stiff segments comparing the effect of the edge-to-edge segment spacing.



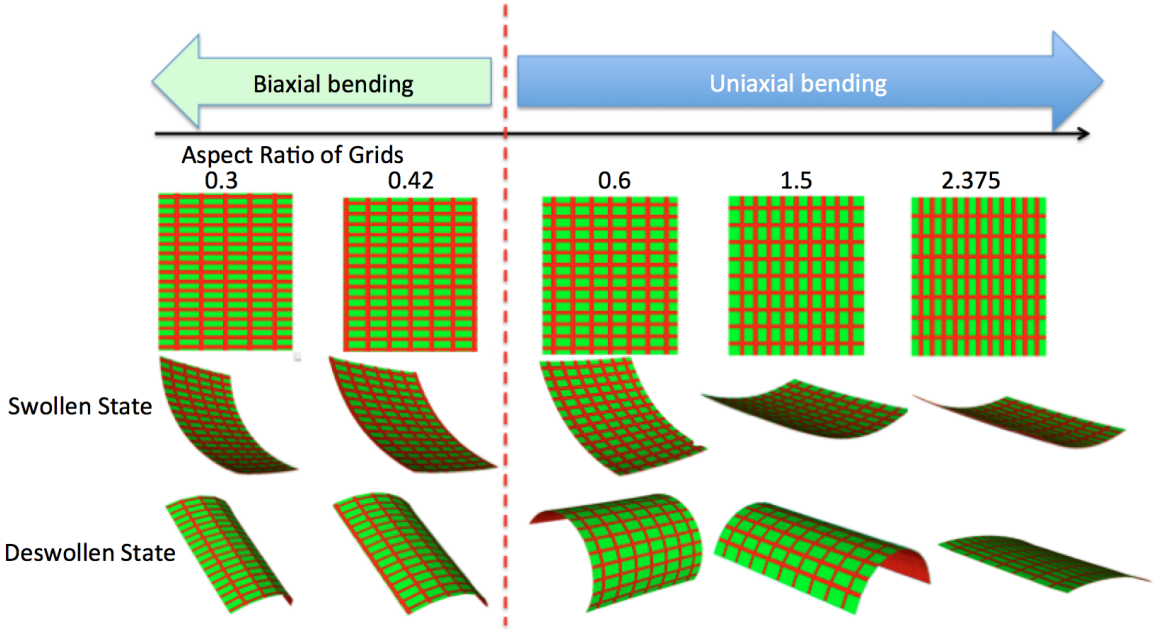
(a) Aspect ratio of the plate: 0.525



(b) Aspect ratio of the plate: 0.757



(c) Aspect ratio of the plate: 0.88



(d) Aspect ratio of the plate: 1.2

Figure 3.5: Deformed configurations for cases with different plate aspect ratio (0.525, 0.757, 0.88 and 1.2) and segment aspect ratio (ranging from 0.3 to 2.65) as summarized in the phase diagram in Fig. 3.4(a). When the segment aspect ratio was close to 1 or when the aspect ratio of the segment was aligned with the plate (i.e., the long axis of the segments were aligned with the long axis of the plate), the plate bent along a single axis, specifically the short axis of the segments. When the segment was aligned perpendicular to the plate and the aspect ratio (long dimension over short dimension) was large, the plate exhibited biaxial bending.

3.3.2 Parametric Study

To explore the design space of the composite structure, we performed a parametric study to examine the effects of geometric factors, including the aspect ratios of the stiff SU-8 segments and of the plate, and the segment spacing, on the biaxial bending behavior. In this parametric study, we considered case 1 with uniform pNIPAM-AAc properties for

simplicity and the thickness of the SU-8 and pNIPAM-AAc are 22 μm and 57 μm respectively, as shown in Fig. 3.1 (c). The aspect ratio of the stiff segments was varied from 0.3 to 2.9 while maintaining the segment spacing and the plate dimensions. This resulted in an irregular row of either shorter or longer SU-8 patches along the plate boundaries. Four sets of simulations were performed, with plate aspect ratio of 0.525, 0.757, 0.88 and 1.2 respectively. The results are summarized in the phase diagram plot of the segment aspect ratio and plate aspect ratio (Fig. 3.4(a)). The deformed configurations in the swollen and deswollen state for all cases are shown in Fig. 3.5. When the segment aspect ratio was close to 1 or when the aspect ratio of the segment was aligned with the plate (i.e., the long axis of the segments were aligned with the long axis of the plate), the plate bent along a single axis, specifically the short axis of the segments. When the segment was aligned perpendicular to the plate and the aspect ratio (long dimension over short dimension) was large, the plate exhibited biaxial bending. For the case with a plate aspect ratio of 0.88, the plate bent biaxially when the segment aspect ratio was greater than 1.2. Similarly a plate with aspect ratio of 1.2 exhibited biaxial bending when the aspect ratio of the SU-8 segments was smaller than 0.42. Recall that the row of stiff segments at the edges of the plate did not have the same aspect ratio as the more interior segments. The irregularly sized patches at the plate edges may have a significant influence on the bending axis near the phase boundary between biaxial and uniaxial bending.

We next varied the segment spacing from 4.4 to 21.9 times the plate thickness while maintaining the segment aspect ratio and plate dimensions. A larger segment spacing led to

a larger curvature and a tighter fold (Fig. 3.4(b)). The plate exhibited wrinkling in addition to bending for large segment spacing. This is demonstrated in Fig. 3.4(c) for a square plate and square segments. The plate exhibited wrinkling when the area covered by stiff segments became less than 70% of the plate area. The same phenomenon was observed experimentally by Hayward et al. [146] for highly crosslinked dots embedded in a lightly crosslinked gel matrix.

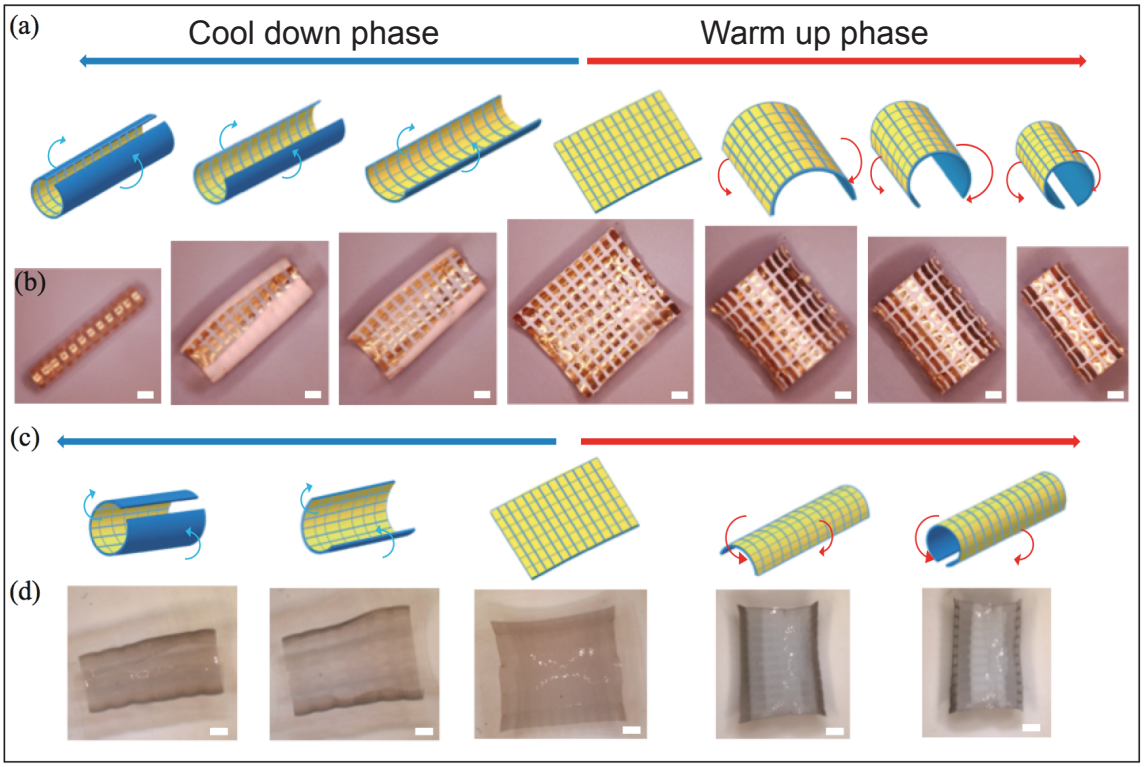


Figure 3.6: Images from actuation experiments exhibiting biaxial bending along with schematic illustrations (a)&(b) First bending sequence; (c)&(d) Second bending sequence. The scale bar indicates 1 mm. A thin coating of gold is dusted on the exposed surface of the stiff SU8 segments in (b) to better visualize the bending axis.

3.3.3 Experiment Results

The results from the temperature-activated bending experiments are summarized in Fig. 3.6. The bilayer plates exhibited biaxial bending as predicted in the simulations. The plates bent up towards positive z during swelling and down, towards negative z during deswelling, which corresponded to case 1 of the simulations with a homogenous pNIPAM-AAc layer. We observed two sequences of biaxial bending in the experiments. One set of specimens bent along the short axis during swelling and the long axis during deswelling, which was in the opposite order of the simulations (Fig. 3.6(b)). The second, smaller set of specimens exhibited the opposite sequence, bending along the long axis during swelling and along the short axis during deswelling, as predicted by the simulations (Fig. 3.6(d)). The bending direction was repeatable in subsequent heating and cooling cycles. However, we observed in some specimens that the curvature progressively decreased with each temperature cycle after the fourth or fifth cycle. We attributed this to the delamination of the stiff SU-8 segments from the pNIPAM-AAc substrate.

While the second biaxial sequence resulted in a lower energy state than the opposite sequence in the modeling study, the energy difference for bending along the short and long axes was small, on the order of 10^{-7}J (Fig. 3.3(c)) for both swelling and deswelling. Many experimental factors may have caused the specimen to exhibit the opposite bending sequence. In the experiments, the solution was heated by a hot plate from the bottom which resulted in a nonuniform temperature field. Heat conduction and solvent diffusion through the plate could cause a nonuniform distribution of swelling ratio within the pNIPAM-AAc,

which may bias the bending direction and axis of the plate. These factors were not included in the simulation for the equilibrium behavior of the plate. In the simulation of case 2, we used two discrete layers with different moduli to approximate the continuous variation in the crosslink density through the plate thickness. This can affect the curvature of the bent composite plate, but should not affect the direction of bending. Furthermore, the simulation started from an idealized, flat, stress free state, while the plate tended to exhibit a saddle shape near the LCST in the experiments.

3.4 Conclusions

We designed a thermoresponsive self-folding system composed of soft swelling gels and stiff non-swelling segments arranged in a regular grid pattern. The self-folding behavior of the composite plate was investigated using experiments and finite element modeling. In both experiments and simulation, the system exhibited bidirectional and biaxial bending. The direction of bending was determined by the distribution of the swelling ratio through the thickness of the composite plate and can be altered by imparting a swelling gradient through the thickness. Biaxial bending occurred when the segments were arranged such that the long axis of the patch was perpendicular to the long axis of the plate. The curvature of the bent plate was controlled by the segment spacing, and a large segment spacing led to the development of wrinkles during bending. The composite structure can be found in many instances in nature, for example in turtle shells and snake skin. The mismatch

strain in these living tissues is provided by growth of the underlying soft dermis. Studying the artificial composite structure will provide a greater understanding of the development of the biological structure as well as design rules for adaptive flexible armor systems.

Chapter 4

An Effective Temperature Model for the Large Strain Hardening Behavior of Glassy Polymers

Amorphous polymers exhibit a viscoplastic strain hardening behavior at large strain. To describe this hardening behavior, we extended the effective temperature theory proposed by Xiao and Nguyen [82] to large strains to incorporate two mechanisms: the stretching and orientation of the polymer network, which leads to the development of a backstress, and network relaxation, which accounts for the temperature and rate dependence of strain hardening. The model was applied to simulate the thermomechanical behavior of polycarbonate (PC) to determine the model parameters from standard thermomechanical tests. The simulation results showed that the model can quantitatively capture the dependence of

the hardening modulus on strain, strain rate, and temperature, as well as the unloading and reloading behavior measured in uniaxial compression tests.

4.1 Introduction

Glassy polymers share an intrinsic stress response that is strongly dependent on temperature, strain rate, time, and deformation history. The material is elastic at small strains, with a Young's modulus that typically ranges between 1 to 3 GPa with little dependence on the temperature and strain rate at temperatures below the onset of the glass transition. At approximately 5% strain, the material yields and plastically deforms when the stress becomes large enough to spur local segmental rearrangements of the polymer chains. The yield stress increases with decreasing temperature, increasing strain rate and annealing time. The temperature-dependence and strain-rate dependence is commonly described using the Eyring model for stress-activated flow [174]. The model relates the stress and plastic strain rate through a nonlinear viscosity that decreases exponentially with temperature and stress when it exceeds an activation stress. Following yield, the stress may decrease with strain to a steady-state draw stress. The yield strength and post-yield stress drop is highly dependent on the thermomechanical history. Annealed specimens exhibit a higher yield strength and more softening than quenched specimens as a consequence of physical aging, while mechanical pre-deformation can mechanically rejuvenate the material and eliminate strain softening [175–178]. A number of phenomenological models have been

developed to describe the physical aging [80, 179]. Xiao and Nguyen showed that the phenomena can be captured by coupling the evolution of the nonequilibrium structure with plastic deformation in an effective temperature thermodynamic framework [82]. In the Xiao and Nguyen theory, the evolution of the nonequilibrium configurational entropy to a more ordered state increases the resistance to plastic deformation, through the nonlinear Adam-Gibbs model [180], which increases the yield strength. Plastic work drives the configurational entropy to a more disordered state, which decreases the resistance to plastic flow and produces plastic strain softening. Following strain softening, the stress increases with strain as a result of long range molecular alignment of the polymer network. The degree of strain hardening relative to the post-yield softening determines the failure behavior of glassy polymers [80, 181]. Polymers with a high strain hardening modulus and small post yield stress drop tend to be ductile, because the strain localization caused by strain softening is suppressed by the increased resistance to plastic deformation caused by molecular orientation [182, 183]. In this work, we aim to extend the Xiao and Nguyen model to describe the temperature-dependent and rate-dependent strain hardening behavior of glassy polymers.

The modeling approach for the strain hardening behavior of glassy polymers can be traced back to the work of Haward and Thackray [184], who first used rubber elasticity theory to describe the post-draw strain hardening. They were inspired by experimental observations that almost all of the plastic deformation can be recovered by heating the material to temperatures above T_g [185–188]. This thermomechanical reversibility is the

underlying mechanism of the shape memory behavior of amorphous polymers. The hardening stress can be derived from the change in network entropy with deformation, which for a one-dimensional case $\sigma = -\lambda T(\frac{\partial s}{\partial \lambda})_T$, where λ is the stretch, s is the entropy per unit volume, σ is the true stress and T is the temperature. Assuming that the chains obey Gaussian statistics yields the Gaussian hardening model

$$\sigma(\lambda) = G_R(\lambda^2 - 1/\lambda), \tag{4.1}$$

where G_R is defined as the hardening modulus. When the end-to-end distance of a polymer chain is stretched near the contour length, a Langevin statistical model is used instead of the Gaussian approximation to describe the dramatic increase in the hardening modulus [189, 190]. The transition between Gaussian hardening and Langevin hardening strongly depends on entanglement length [191]. Boyce et al. [192, 193] extended the one dimensional constitutive equation proposed by Haward and Thackray to a three dimensional formulation known as the 'BPA-model'. The spatial distribution of molecular chains in the BPA-model was initially represented by the 'three-chain' model [194] and later the more predictive 'eight-chain' model [190] and the 'full-chain' model by Wu and Van Der Giessen [195].

While this approach can capture the hardening stress response under certain temperature and strain rate for various semi-crystalline polymers [196–198] and amorphous glasses [199, 200], significant inconsistencies have been revealed through experimental observa-

tions. Experiments show that the strain hardening modulus is smaller at higher temperatures [191, 201–206], while in the entropic model the hardening modulus increases with temperature. Moreover, the hardening modulus predicted from the entropic models is orders of magnitude smaller than that measured in experiments [184, 204, 207]. In addition, experiments [16, 208, 209] and molecular dynamics simulations [210–212] show that the internal energy contributes significantly to strain hardening, which is not included in entropic network models. Hoy and Robbins [191, 213] revealed that there is a dramatic increase in energetic stress as polymer chains are stretched taut between entanglements. Given these discrepancies, alternative approaches have been developed in recent years to model the strain hardening behavior. Chen and Schweizer [214] attributed the strain hardening to the strengthened activation barrier and an increase in viscosity as a result of the deformation induced anisotropy in chain conformations. This idea has been applied to the constitutive modeling by introducing a deformation dependence in the viscous flow stress [17, 18, 215–218]. Anand et al. [219, 220] applied the internal variable approach by introducing macroscopic internal variables to represent the microstructural resistance to plastic flow.

Hansan and Boyce [16] performed differential scanning calorimetric measurements of cold worked specimens and their observation of two distinct and separately evolving exothermic undershoots in the DSC curves suggest the presence of two independent relaxation mechanisms in glassy polymers. Dupaix and Boyce [221] developed a constitutive model that describes the amorphous polymers with two parallel Maxwell elements that

represent the resistance to local segmental motion and the resistance to long range network deformation. A molecular relaxation mechanism was incorporated into the network resistance to capture more accurately the temperature and strain rate dependence of strain hardening. Hempel [222] incorporated the molecular relaxation mechanism in the kinematic hardening approach. The network relaxation was assumed to be in series with the stretching and orientation of polymer chains, leading to temperature and strain rate dependent backstress. Other works incorporating molecular relaxation mechanism in the modeling of thermoplastics as well as rubbers include [223–226].

Xiao and Nguyen [82, 227] developed a coupled thermomechanical theory, based on the effective temperature thermodyndamic framework introduced by Nieuwenhuizen et al. [228] to describe the thermomechanical coupled behavior of amorphous polymers. The theory assumes that the thermodynamic behavior of glassy material can be described by two weakly interacting material subsystems: the kinetic/vibrational subsystem, characterized by the kinetic entropy and temperature, and a configurational subsystem characterized by the configurational entropy and a higher effective temperature. The effective temperature, which represents the nonequilibrium configurational structure, evolves towards the equilibrium temperature and the resulting decrease in the configurational entropy increases the resistance to plastic deformation, which increases the yield stress. Plastic work increases the configurational entropy, which decreases the resistance to plastic deformation leading to strain-softening and the post-yield stress drop. The model was able to accurately describe the thermomechanical behavior of amorphous polymers at moderate strain levels

after yield. To extend the Xiao and Nguyen theory to strain hardening, we incorporated a backstress to describe the energy stored by the long-range deformation of the polymer network and a temperature-dependent molecular relaxation mechanism to describe the rate-dependence and temperature-dependence of strain hardening [221, 222]. The constitutive model was applied to simulate the uniaxial compression of polycarbonate (PC). Results showed that the model was able to accurately capture the temperature-dependence and rate-dependence of strain hardening for polycarbonate.

4.2 Method

4.2.1 Experimental methods

4.2.1.1 Materials and specimen preparation

Polycarbonate (PC) filaments with a molecular weight of $M_n = 1.671 \times 10^4$ g/mol and a filament diameter of 2.85 ± 0.05 mm were purchased from Ultimaker (Watermolenweg 2, 4191PN, Geldermalsen, The Netherlands). The filaments were cut into cylindrical specimens with an average length of 4 mm for uniaxial compression tests. The cut surfaces were polished with sandpaper to obtain a smooth surface. Thin film specimens used for dynamic frequency sweep tests were made by melting the PC filaments at 200°C and injecting the fluid into a square steel mold with a length of 50.0 mm and a depth of 2.0 mm and cooling down to room temperature. The film was cut into 20.0 mm $\times$ 5.0 mm $\times$ 2.0 mm specimens

with a band saw. Specimens were stored with desiccants to avoid absorption of water.

4.2.1.2 Dynamic frequency sweep tests

We used a method developed previously [82, 229, 230] to characterize the stress relaxation spectrum of the PC material. Dynamic frequency sweep tests were performed using a TA Q800 Dynamic Mechanical Analyzer (DMA). Thin film specimens with dimensions specified in Sec. 4.2.1.1 were mounted between the tensile grips with a gauge length of 10 mm. The specimens were subjected to stepwise temperature increase from 35°C to 143°C in 2°C increments. At each temperature step, a dynamic strain of 0.2% amplitude was applied at 0.32 Hz, 1.0 Hz, 3.2 Hz, 10.0 Hz, and 31.6 Hz frequencies to measure the storage modulus $G(\omega, T)$. Applying the time-temperature superposition (TTS) principle for thermorheologically simple materials, the storage moduli at different temperatures were shifted to the reference temperature $T_0 = 143$°C to obtain the master curve of the frequency response [231]. The resulting temperature dependent shift factor $a(T)$ in the glass transition temperature range was used to determine the parameters of the Williams-Landel-Ferry (WLF) equation,

$$\log a_T(T) = \frac{-C_1^0(T - T_0)}{C_2^0 + T - T_0}, \tag{4.2}$$

where C_1^0 and C_2^0 are the WLF constants at the reference temperature T_0. The WLF constants can be shifted to T_g as $C_2^g = C_2^0 + T_g$ and $C_1^g C_2^g = C_1^0 C_2^0$ [231].

4.2.1.3 Differential scanning calorimetry

DSC measurements were performed to determine the heat capacity of the rubbery and glassy material and the structural relaxation spectrum. Specimens weighing 8.9 mg were cut from the PC filament and put in a TA Q20 DSC. The specimens were equilibrated at 150°C for 30 mins to remove the thermal history then cooled to 88°C at 3°C/min and annealed for 24 hours. The annealed specimens were cooled to 20°C at 3°C/min and then heated to 250°C at 5°C/min to measure the heat flow rate.

4.2.1.4 Uniaxial compression

Uniaxial compression tests at constant temperatures were performed to measure the strain hardening behavior of polycarbonate. Cylindrical specimens with dimensions specified in Sec. 4.2.1.1 were equilibrated at 143°C and annealed 30 mins to erase the thermal history. The specimens were cooled at 3°C/min to the test temperatures, 95°C, 37°C, 27°C or 15°C and annealed at the test temperature for 30 mins in an incubator (Ecotherm, Hartkirchen, Austria). The annealed specimens were placed in an MTS Insight 5 electromechanical testing system (MTS, Eden Prairie, MN, USA) equipped with an environmental temperature chamber (Thermcraft, Inc. Winston Salem, NC, USA) set at the test temperature, and subjected to compressive loading to 100% true strain at true strain rates 10^{-3}/s, 10^{-4}/s, 10^{-5}/s. Three tests were performed at each strain rate and temperature to ensure repeatability. To measure the material response to unloading and reloading, the specimens were first compressed at room temperature (27°C) to 60% true strain at an engineering

strain rate of 10^{-3}/s or 10^{-4}/s, unloaded to zero force at the same strain rate and reloaded to 100% true strain at the same strain rate.

4.2.2 Modeling methods

4.2.2.1 A one-dimensional small-deformation model

We consider a simple one-dimensional case with small deformation first. A linear rheological model is shown in Fig. 4.1(a) as a schematic representation. In the rheological model, the spring with shear modulus of μ^{eq} describes the rubbery stress response at high temperatures. For thermoplastic materials $\mu^{eq} = 0$ because of the absence of chemical crosslink. The total deformation was decomposed to an elastic component represented by a spring with shear modulus of μ^{neq}, and a viscous component represented by a dashpot with characteristic relaxation time of τ_S,

$$\varepsilon = \varepsilon^e + \varepsilon^v. \tag{4.3}$$

The elastic component describes the resistance of local segmental interaction while the viscous component describes the segmental relaxation, as discussed in [122, 229, 230]. To model the temperature and rate dependent strain hardening behavior of glassy polymers, we adopted the same assumption as Hempel [222] that the inelastic deformation can be further decomposed to a network elastic component represented by a spring with shear modulus of μ^{back}, and a network viscous component represented by a dashpot with characteristic

relaxation time of τ_N,

$$\varepsilon^v = \varepsilon^{N,e} + \varepsilon^{N,v}. \tag{4.4}$$

The network elastic component describes the resistance to long-range network deformation coming from the stretching and orientation of polymer chains while the network viscous component describes the long-range molecular relaxation [221, 222]. Compared to a parallel arrangement between the resistance to local segmental motion and the resistance to long range network deformation that is numerically simple, this decomposition incorporates the molecular relaxation mechanism in the kinematic hardening approach. The kinematic hardening indicates a shift of the yield surface, which allows the model to capture phenomena related to unloading and reloading that were observed in experiments, such as the Bauschinger effect [17, 18] and the deformation induced anisotropy [19].

The free energy density can be additively split to an equilibrium component describing the rubbery response at high temperatures, a nonequilibrium component describing the stiff glassy response at low temperatures and a network component that represents the energy stored in the resistance to long-range network deformation,

$$\Psi = \frac{1}{2}E^{eq}\varepsilon^2 + \frac{1}{2}E^{neq}\varepsilon^{e^2} + \frac{1}{2}E^{back}\varepsilon^{N,e^2}. \tag{4.5}$$

The stress is given by $\sigma = \frac{\partial \Psi}{\partial \varepsilon}$ and can be expressed as the sum of equilibrium and nonequilibrium contributions,

$$\sigma = \underbrace{E^{eq}\varepsilon}_{\sigma^{eq}} + \underbrace{E^{neq}\varepsilon^e}_{\sigma^{neq}}, \tag{4.6}$$

and the backstress is given by

$$\sigma^{back} = E^{back}\varepsilon^{N,e}. \tag{4.7}$$

The viscous strains ε^{v} and $\varepsilon^{N,v}$ evolve under their respective flow rules, which can be expressed as,

$$\dot{\varepsilon^{v}} = \frac{1}{\nu^{I}}(\sigma^{neq} - \sigma^{back}), \tag{4.8}$$

$$\dot{\varepsilon^{N,v}} = \frac{1}{\nu^{N}}\sigma^{back}, \tag{4.9}$$

where ν^{I} and ν^{N} are the flow resistances to segmental rearrangement and network deformation respectively.

4.2.2.2 Kinematics

The model in above section can be extended to three-dimensional finite-deformation. Consider a deformation map, $\mathbf{x} = \varphi(\mathbf{X})$, which carries points $\mathbf{X}$ in the reference configuration to points $\mathbf{x}$ in the spatial configuration. The deformation gradient is defined as $\mathbf{F} = \partial\varphi/\partial\mathbf{X}$. To account for the broad relaxation spectrum, the single-process model shown in Fig. 4.1(a) was extended to multi-process with P parallel components. Instead of an additive split in eq. (4.3) for one-dimensional model, the total deformation gradient $\mathbf{F}$ was split multiplicatively into elastic and viscous components,

$$\mathbf{F} = \mathbf{F}_{k}^{e}\mathbf{F}_{k}^{v}, \quad \text{for} \quad k = 1...P. \tag{4.10}$$

The effect of thermal deformation is not considered here for simplicity. The right and left Cauchy-Green deformation tensors for $\mathbf{F}$ and its elastic components are defined as,

$$\begin{aligned} &\mathbf{C} = \mathbf{F}^T\mathbf{F}, \mathbf{b} = \mathbf{F}\mathbf{F}^T, \\ &\mathbf{C}_k^e = \mathbf{F}_k^{e^T}\mathbf{F}_k^e, \mathbf{b}_k^e = \mathbf{F}_k^e\mathbf{F}_k^{e^T}, \quad \text{for} \quad k = 1...P, \end{aligned} \tag{4.11}$$

and $\mathbf{L} = \dot{\mathbf{F}}\mathbf{F}^{-1}$ and $\mathbf{L}_k^v = \dot{\mathbf{F}}_k^v\mathbf{F}_k^{v^{-1}}$ are the total and inelastic spatial velocity gradients. Analogous to eq. (4.4), the inelastic deformation gradient $\mathbf{F}_k^v$ was decomposed into two components to describe the network deformation and relaxation as,

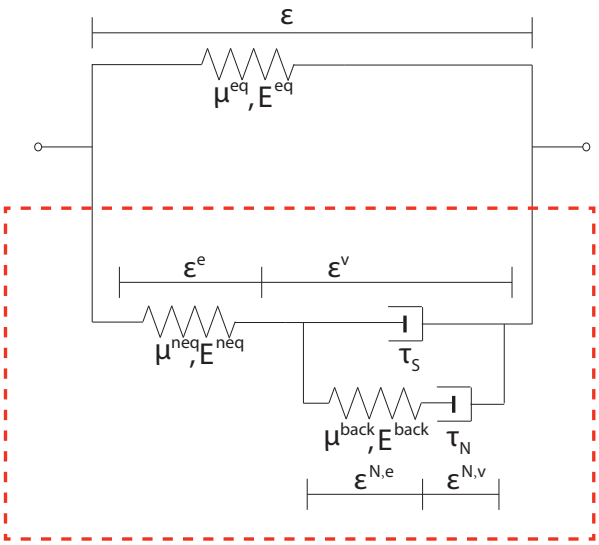
$$\mathbf{F}_k^v = \mathbf{F}_k^{N,e}\mathbf{F}_k^{N,v}, \quad \text{for} \quad k = 1...P. \tag{4.12}$$

where $\mathbf{F}_k^{N,e}$ describes the orientation and stretching of the polymer chains and $\mathbf{F}_k^{N,v}$ represents the molecular relaxation [221, 222]. The right and left Cauchy-Green deformation tensors for the network elastic component can be defined in a similar way,

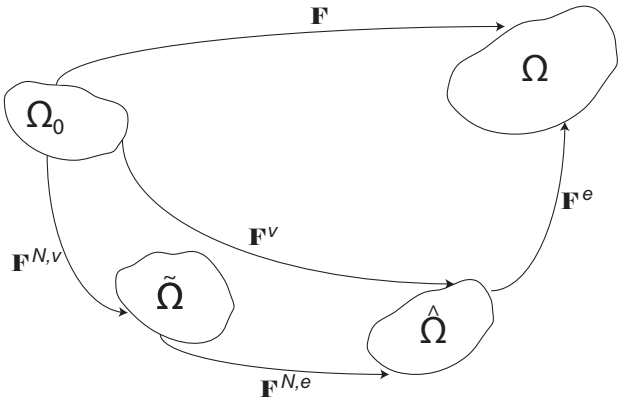
$$\mathbf{C}_k^{N,e} = \mathbf{F}_k^{N,e^T}\mathbf{F}_k^{N,e}, \mathbf{b}_k^{N,e} = \mathbf{F}_k^{N,e}\mathbf{F}_k^{N,e^T}, \quad \text{for} \quad k = 1...P, \tag{4.13}$$

and the network elastic and inelastic velocity gradient is define as $\mathbf{L}_k^{N,e} = \dot{\mathbf{F}}_k^{N,e}\mathbf{F}_k^{N,e^{-1}}$ and $\mathbf{L}_k^{N,v} = \mathbf{F}_k^{N,e}\dot{\mathbf{F}}_k^{N,v}\mathbf{F}_k^{N,v^{-1}}\mathbf{F}_k^{N,e^{-1}}$. Note that $\mathbf{L}_k^v = \mathbf{L}_k^{N,e} + \mathbf{L}_k^{N,v}$. The sequence of deformation maps generated from the successive decompositions of the deformation gradient is illustrated in Fig. 4.1(b). The decomposition of the inelastic deformation gradient in eq. (4.12)

introduced an additional intermediate configuration $\widetilde{\Omega}$, as shown in the deformation map in Fig. 4.1(b).



(a)



(b)

Figure 4.1: (a) A single-process rheological representation of the model. The elements in the red dashed box were repeated for a multi-process model. (b) Decomposition scheme for the deformation gradient.

4.2.2.3 Effective temperature nonequilibrium thermodynamic framework for viscoplasticity

In this section we derive the general effective temperature thermodynamic framework for the large deformation viscoplastic behavior of polymers. The basic idea of the effective temperature theory is that the behavior of amorphous polymers can be characterized by a kinetic process that equilibrates instantaneously and Q configurational modes that gradually evolve to equilibrium, characterized by a series of state variables T_{e_i} known as the spectrum of effective temperatures [82]. The first law of thermodynamics in the reference configuration can be written as,

$$\dot{e} = \mathbf{S} : \frac{1}{2}\dot{\mathbf{C}} - \nabla_{\mathbf{X}}.\mathbf{Q}, \tag{4.14}$$

where $\mathbf{S}$ is the second Piola-Kirchhoff stress tensor, e is the internal energy density and $\mathbf{Q}$ is the heat flux. The second law of thermodynamics can be written in the reference configuration as,

$$\dot{\eta} + \nabla_{\mathbf{X}}.\mathbf{H} \geq 0, \tag{4.15}$$

where η is the entropy density and $\mathbf{H}$ is the entropy flux. We assume that the internal energy, total entropy, entropy flux and heat flux can be decomposed into the kinetic and

configurational parts as,

$$\begin{aligned} e &= e^k + \sum_i^Q e_i^c, \eta = \eta^k + \sum_i^Q \eta_i^c, \\ \mathbf{H} &= \mathbf{H}^k + \sum_i^Q \mathbf{H}_i^c, \mathbf{Q} = \mathbf{Q}^k + \sum_i^Q \mathbf{Q}_i^c. \end{aligned} \tag{4.16}$$

The kinetic and configurational entropy flux are defined as [232],

$$\mathbf{H}^k = \frac{\mathbf{Q}^k}{T}, \mathbf{H}_i^c = \frac{\mathbf{Q}_i^c}{T_{e_i}}. \tag{4.17}$$

We expand the second law by applying the decompositions in eqs. (4.16) and (4.17),

$$\dot{\eta^k} + \sum_i^Q \dot{\eta_i^c} + \frac{\nabla_{\mathbf{X}}.\mathbf{Q}^k}{T} + \sum_i^Q \frac{\nabla_{\mathbf{X}}.\mathbf{Q}_i^c}{T_{e_i}} - \frac{1}{T^2}\mathbf{Q}^k.\nabla_{\mathbf{X}}T - \sum_i^Q \frac{1}{{T_{e_i}}^2}\mathbf{Q}_i^c.\nabla_{\mathbf{X}}T_{e_i} \geq 0. \tag{4.18}$$

The Helmholtz free energy density is defined as [228],

$$\Psi = \Psi^k + \sum_i^Q \Psi_i^c = e^k - T\eta^k + \sum_i^Q (e_i^c - T_{e_i}\eta_i^c). \tag{4.19}$$

The free energy density Ψ depends on independent state variables of temperature, effective temperatures, total deformation, and internal variables including the elastic deformation tensors $\mathbf{C}_k^e$ and the elastic network deformation tensors $\mathbf{b}_k^{N,e}$ as $\Psi(T, T_{e_1}...T_{e_Q}, \mathbf{C}, \mathbf{C}_1^e...\mathbf{C}_P^e, \mathbf{b}_1^{N,e}...\mathbf{b}_P^{N,e})$. Substituting eq. (4.19) in eq. (4.14) and rewriting the first law using the Helmholtz free

energy density gives,

$$T\,\dot{\eta^k} + \sum_i^Q T_{e_i}\,\dot{\eta}_i^c + \nabla_{\mathbf{X}}.\mathbf{Q}^k + \sum_i^Q \nabla_{\mathbf{X}}.\mathbf{Q}_i^c = \mathbf{S} : \frac{1}{2}\dot{\mathbf{C}} - \dot{T}\,\eta^k - \sum_i^Q \dot{T}_{e_i}\,\eta_i^c - \dot{\Psi}. \tag{4.20}$$

The above equation is applied to eliminate the kinetic entropy and heat flux in eq. (4.18),

$$\begin{aligned}&\sum_i^Q (T - T_{e_i})\,\dot{\eta}_i^c + \sum_i^Q (T - T_{e_i})\frac{\nabla_{\mathbf{X}}.\mathbf{Q}_i^c}{T_{e_i}} - \frac{1}{T}\mathbf{Q}^k.\nabla_{\mathbf{X}}T - \sum_i^Q \frac{T}{{T_{e_i}}^2}\mathbf{Q}_i^c.\nabla_{\mathbf{X}}T_{e_i} \\ &+ \mathbf{S} : \frac{1}{2}\dot{\mathbf{C}} - \dot{T}\,\eta^k - \sum_i^Q \dot{T}_{e_i}\,\eta_i^c - \dot{\Psi} \geq 0.\end{aligned} \tag{4.21}$$

The rate of change of free energy density can be evaluated as,

$$\dot{\Psi} = \frac{\partial \Psi}{\partial T}\dot{T} + \sum_i^Q \frac{\partial \Psi}{\partial T_{e_i}}\dot{T}_{e_i} + 2\frac{\partial \Psi}{\partial \mathbf{C}} : \frac{1}{2}\dot{\mathbf{C}} + \sum_j^P \frac{\partial \Psi}{\partial \mathbf{C}_j^e} : \dot{\mathbf{C}_j^e} + \sum_j^P \frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}} : \dot{\mathbf{b}_j^{N,e}}. \tag{4.22}$$

It can be shown that $\frac{\partial \Psi}{\partial \mathbf{C}_j^e} : \dot{\mathbf{C}_j^e} = \frac{\partial \Psi}{\partial \mathbf{F}_j^v} : \dot{\mathbf{F}_j^v} = -2\mathbf{C}_j^e \frac{\partial \Psi}{\partial \mathbf{C}_j^e} : \mathbf{L}_j^v$. Substituting eq. (4.22) into the second law in eq. (4.21) gives,

$$\begin{aligned}&\sum_i^Q (T - T_{e_i})\,\dot{\eta}_i^c + \sum_i^Q (T - T_{e_i})\frac{\nabla_{\mathbf{X}}.\mathbf{Q}_i^c}{T_{e_i}} - \frac{1}{T}\mathbf{Q}^k.\nabla_{\mathbf{X}}T - \sum_i^Q \frac{T}{{T_{e_i}}^2}\mathbf{Q}_i^c.\nabla_{\mathbf{X}}T_{e_i} \\ &+ (\mathbf{S} - 2\frac{\partial \Psi}{\partial \mathbf{C}}) : \frac{1}{2}\dot{\mathbf{C}} - (\eta^k + \frac{\partial \Psi}{\partial T})\,\dot{T} - \sum_i^Q (\eta^c{}_i + \frac{\partial \Psi}{\partial T_{e_i}})\,\dot{T}_{e_i} + \underbrace{2\sum_j^P \mathbf{C}_j{}^e \frac{\partial \Psi}{\partial \mathbf{C}_j^e} : \mathbf{L}_j^v}_{W^{\mathrm{I}}} - \underbrace{\sum_j^P \frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}} : \dot{\mathbf{b}_j^{N,e}}}_{W^N} \geq 0,\end{aligned} \tag{4.23}$$

where W^I and W^N represents the internal dissipation from inelastic deformation from local segmental motions and network deformation respectively. Consider an arbitrary approach

to equilibrium, $\mathbf{S}$, η_k and ${\eta^c}_i$ must equal the following to satisfy the above inequality [233],

$$\mathbf{S} = 2\frac{\partial \Psi}{\partial \mathbf{C}}, \eta_k = -\frac{\partial \Psi}{\partial T}, {\eta^c}_i = -\frac{\partial \Psi}{\partial T_{e_i}}. \tag{4.24}$$

Applying the above constitutive relations to eq. (4.23) yields a reduced dissipation inequality,

$$\sum_i^Q (T - T_{e_i})\,\dot{\eta}_i^c + \sum_i^Q (T - T_{e_i})\frac{\nabla_{\mathbf{X}}.\mathbf{Q}_i^c}{T_{e_i}} - \frac{1}{T}\mathbf{Q}^k.\nabla_{\mathbf{X}}T - \sum_i^Q \frac{T}{{T_{e_i}}^2}\mathbf{Q}_i^c.\nabla_{\mathbf{X}}T_{e_i} + W^I - W^N \geq 0. \tag{4.25}$$

The above equation restricted the entropy production from inelastic deformation and heat conduction. Substituting eqs. (4.22) and (4.24) into the first law in eq. (4.20) gives,

$$T\,\dot{\eta}^k + \sum_i^Q T_{e_i}\,\dot{\eta}_i^c = -\nabla_{\mathbf{X}}.\mathbf{Q}^k - \sum_i^Q \nabla_{\mathbf{X}}.\mathbf{Q}_i^c + W^{I,k} + \sum_i^Q W_i^{I,c} - W^N, \tag{4.26}$$

where W^I was decomposed into kinetic and configurational contributions, $W^{I,k} = 2\sum_j^P {\mathbf{C}_j}^e \frac{\partial \Psi^k}{\partial \mathbf{C}_j^e} : \mathbf{L}_j^v$ and $W_i^{I,c} = 2\sum_j^P {\mathbf{C}_j}^e \frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} : \mathbf{L}_j^v$ based on the decomposition of Ψ in eq. (4.19). We make the same assumptions as [82] to simplify the model: (1) The interactions between Q configurational subsystems are negligible. (2) The configurational entropy of a certain subsystem only depends on the effective temperature of that specific subsystem, not on any other effective temperatures. Moreover, we assume that the contributions of the network resistance arise from the kinetic material subsystem. The last assumption is discussed further in Sec. 4.6. With these assumptions, we can decompose the first law into energy balances for

each subsystem as,

$$\begin{aligned} T\,\dot{\eta}^k &= -\nabla_{\mathbf{X}}.\mathbf{Q}^k - \sum_i^Q Q_i^{kc} + W^{I,k} - W^N, \\ T_{e_i}\,\dot{\eta}_i^c &= -\nabla_{\mathbf{X}}.\mathbf{Q}_i^c + Q_i^{kc} + W_i^{I,c}, \text{for} i = 1...Q. \end{aligned} \tag{4.27}$$

In addition, the internal power from the network resistance can be evaluated as,

$$W_N = \sum_j^P \frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}} : \dot{\mathbf{b}}_j^{N,e} = \sum_j^P 2\frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}}\mathbf{b}_j^{N,e} : \mathbf{L}_j^{N,e} = \sum_j^P 2\frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}}\mathbf{b}_j^{N,e} : (\mathbf{L}_j^v - \mathbf{L}_j^{N,v}). \tag{4.28}$$

Substitute eqs. (4.27) and (4.28) in the second law (4.25) to eliminate the configurational heat flux gives,

$$\begin{aligned} &\sum_i^Q -(T_{e_i} - T)\frac{Q_i^{kc}}{T_{e_i}} - \frac{1}{T}\mathbf{Q}^k.\nabla_{\mathbf{X}}T - \sum_i^Q \frac{T}{{T_{e_i}}^2}\mathbf{Q}_i^c.\nabla_{\mathbf{X}}T_{e_i} \\ &+ \sum_j^P 2(\sum_i^Q \frac{T}{T_{e_i}}\mathbf{C}_j^e\frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} + \mathbf{C}_j^e\frac{\partial \Psi^k}{\partial \mathbf{C}_j^e} - \frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}}\mathbf{b}_j^{N,e}) : \mathbf{L}_j^v + \sum_j^P 2\frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}}\mathbf{b}_j^{N,e} : \mathbf{L}_j^{N,v} \geq 0, \end{aligned} \tag{4.29}$$

For a special case where $T_{e_i} = T$ and $\mathbf{L}_j^v = \mathbf{L}_j^{N,v} = \mathbf{0}$, only the heat conduction terms survive in the inequality (4.29). The inequalities of each subsystem, $-\mathbf{Q}^k \cdot \nabla_{\mathbf{X}}T \geq 0$ and $-\mathbf{Q}_i^c \cdot \nabla_{\mathbf{X}}T_{e_i} \geq 0$, can be satisfied by a Fourier-type conduction law for each subsystem,

$$\mathbf{q}^k = -k^k\nabla_{\mathbf{x}}T, \quad \mathbf{q}_i^c = -k_i^c\nabla_{\mathbf{x}}T_{e_i}, \tag{4.30}$$

where $\mathbf{q}^k = 1/\det(\mathbf{F})\mathbf{F}\mathbf{Q}^k$ and $\mathbf{q}_i^c = 1/\det(\mathbf{F})\mathbf{F}\mathbf{Q}_i^c$ are the kinetic and configurational contributions to the spatial heat flux, and $k^k \geq 0$ and $k_i^c \geq 0$ are the thermal conductivity of

each subsystem.

Given that the inequality in heat conduction has been satisfied, removing these terms from eq. (4.29) provides a stronger constraint for the entropy production from the internal power of inelastic deformation and network resistance, and from the inter-subsystem heat exchange. The constraint can be satisfied by requiring each term to be non-negative. For this purpose we define the following stress tensors,

$$\begin{aligned}
\mathbf{M}_j &= 2(\sum_i^Q \frac{T}{T_{e_i}} \mathbf{C}_j^e \frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} + \mathbf{C}_j^e \frac{\partial \Psi^k}{\partial \mathbf{C}_j^e}), \\
\mathbf{N}_j{}^{back} &= 2\frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}} \mathbf{b}_j^{N,e}, \\
\zeta_j &= \mathbf{M}_j - \mathbf{N}_j{}^{back},
\end{aligned} \tag{4.31}$$

where $\mathbf{M}_j$ is the Mandel stress tensor. $\mathbf{N}_j{}^{back}$ is the backstress generated from the network deformation and ζ_j, represents the driving stress for viscoplastic deformation. It can be shown that for an isotropic material, where Ψ is a function of the invariants of the total and inelastic deformation tensors, $\mathbf{M}_j$, $\mathbf{N}_j{}^{back}$ and ζ_j are symmetric, which allows the non-negative dissipation criteria for internal power and internal heat conduction to be written as,

$$\begin{aligned}
&\zeta_j : \mathbf{D}_j^v \geq 0, \\
&\mathbf{N}_j{}^{back} : \mathbf{D}_j^{N,v} \geq 0, \\
&-Q_i^{kc}(T_{e_i} - T) \geq 0,
\end{aligned} \tag{4.32}$$

where $\mathbf{D}_j^v = \frac{1}{2}(\mathbf{L}_j^v + \mathbf{L}_j^{v^T})$ and $\mathbf{D}_j^{N,v} = \frac{1}{2}(\mathbf{L}_j^{N,v} + \mathbf{L}_j^{N,v^T})$ represent the inelastic and the network relaxation rate of deformation tensor respectively. The following constitutive rela-

tions satisfy above inequalities automatically:

$$\begin{aligned} \mathbf{D}_j^{\nu} &= \tfrac{1}{2\nu_j^I}\zeta_j \\ \mathbf{D}_j^{N,\nu} &= \tfrac{1}{2\nu_j^N}\mathbf{N}_j{}^{back}, \quad \text{for} \quad j = 1...P, \end{aligned} \tag{4.33}$$

$$Q_i^{kc} = -K_i(T_{e_i} - T), \quad \text{for} \quad i = 1...Q, \tag{4.34}$$

where $\nu_j^I > 0$, $\nu_j^N > 0$ and $K_i > 0$.

Substituting the constitutive relations in eqs. (4.30) and (4.34) in eq. (4.27) gives the evolution equations for the temperature and effective temperatures,

$$\begin{aligned} c_g \dot{T} = &\sum_i^Q K_i(T_{e_i} - T) + \sum_j^P 2\mathbf{C}_j^e \frac{\partial \Psi^k}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^{\nu} - T 2\frac{\partial \eta^k}{\partial \mathbf{C}} : \frac{1}{2}\dot{\mathbf{C}} \\ &+ T\sum_j^P \mathbf{C}_j^e \frac{\partial \eta^k}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^{\nu} + k^k \nabla_{\mathbf{X}} . (\mathbf{C}^{-1}\nabla_{\mathbf{X}} T) - \sum_j^P 2\frac{\partial \Psi}{\partial \mathbf{b}_j^{N,e}}\mathbf{b}_j^{N,e} : \mathbf{D}_j^{N,e}, \end{aligned} \tag{4.35}$$

$$\begin{aligned} \Delta c_i \dot{T}_{e_i} = &-K_i(T_{e_i} - T) + \sum_j^P 2\mathbf{C}_j^e \frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^{\nu} - T_{e_i} 2\frac{\partial \eta_i^c}{\partial \mathbf{C}} : \frac{1}{2}\dot{\mathbf{C}} \\ &+ T_{e_i}\sum_j^P \mathbf{C}_j^e \frac{\partial \eta_i^c}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^{\nu} + k_i^c \nabla_{\mathbf{X}} . (\mathbf{C}^{-1}\nabla_{\mathbf{X}} T_{e_i}), \end{aligned} \tag{4.36}$$

where $c_g = -T\frac{\partial^2 \Psi}{\partial T^2}$ is the heat capacity of the kinetic subsystem and $\Delta c_i = -T_{e_i}\frac{\partial^2 \Psi}{\partial T_{e_i}{}^2}$ are the heat capacities of the configurational subsystems. Dividing eq. (4.36) by Δc_i gives the

evolution equation for effective temperatures,

$$\begin{aligned}\dot{T_{e_i}} = &-\frac{1}{\tau_{R_i}}(T_{e_i}-T)+\frac{1}{\Delta c_i}\sum_j^P 2\mathbf{C}_j^e\frac{\partial\Psi_i^c}{\partial\mathbf{C}_j^e}:\mathbf{D}_j^v-\frac{T_{e_i}}{\Delta c_i}2\frac{\partial\eta_i^c}{\partial\mathbf{C}}:\frac{1}{2}\dot{\mathbf{C}}\\&+\frac{T_{e_i}}{\Delta c_i}\sum_j^P\mathbf{C}_j^e\frac{\partial\eta_i^c}{\partial\mathbf{C}_j^e}:\mathbf{D}_j^v+\frac{k_i^c}{\Delta c_i}\nabla_{\mathbf{X}}.(\mathbf{C}^{-1}\nabla_{\mathbf{X}}T_{e_i}),\end{aligned}\tag{4.37}$$

where $\tau_{R_i}=\frac{K_i}{\Delta c_i}$ is the structural relaxation time that characterize the rate of evolution towards the equilibrium state. Note that eq. (4.37) is identical to the evolution equation for T_{e_i} in the Xiao and Nguyen model [82] because of the assumption that the stored energy of network deformation is part of the kinetic subsystem and not the configurational subsystem. Xiao and Nguyen [82] discussed the physical significance of eq. (4.37) in detail. The first term on the right hand side describes structural relaxation which drives the configurational subsystems equilibrium. Conversely, the second term represents the mechanical rejuvenation that drives configurational subsystems away from equilibrium. The third and fourth term are thermomechanical coupling terms while the last term represents the diffusion of the effective temperatures.

4.2.2.4 Constitutive model

In this section we develop a constitutive model using the thermodynamic framework. The quasi-incompressible Neo-Hookean model is applied for the internal energy and entropy densities associated with local segmental motion and the Arruda-Boyce eight-chain model [190] is applied for the internal energy associated with network deformation to ac-

count for the limiting stretch of the network structure:

$$\Psi^k = \sum_j^P \frac{(1-a)\mu_j^{neq}}{2}(\mathrm{tr}(\bar{\mathbf{C}}_j^e)-3) + \frac{\kappa}{4}(J^2 - 2\log J - 1)$$
$$+ c_{g0}(T-T_0) - c_{g0}T\ln\frac{T}{T_0} - \frac{c_{g1}}{2}(T-T_0)^2 + \sum_j^P \xi(\overline{\mathbf{b}_j^{N,e}}),$$
$$\Psi_i^c = \frac{T_{e_i}}{T_0}\frac{\mu_i^{eq}}{2}(\mathrm{tr}(\bar{\mathbf{C}})-3) + \sum_j^P \frac{a_i\mu_j^{neq}}{2}(\mathrm{tr}(\bar{\mathbf{C}}_j^e)-3) + \Delta c_{0_i}(T_{e_i}-T_0) - \Delta c_{0_i}T_{e_i}\ln\frac{T_{e_i}}{T_2} - \frac{\Delta c_{1_i}}{2}(T_{e_i}-T_2)^2, \tag{4.38}$$

$$\xi(\overline{\mathbf{b}_j^{N,e}}) = \mu_j^{back}\lambda_L^2[\frac{\bar{\lambda}_{eff,j}}{\lambda_L}x_j + \ln(\frac{x_j}{\sinh x_j})],$$
$$x_j = \mathscr{L}^{-1}(\frac{\bar{\lambda}_{eff,j}}{\lambda_L}), \tag{4.39}$$
$$\bar{\lambda}_{eff,j} = \sqrt{\tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_j^{N,e}})},$$

where $J = \det\mathbf{F}$ is the volumetric deformation. The $\overline{\mathbf{C}} = J^{-2/3}\mathbf{C}$, $\overline{\mathbf{C}_j^e} = J_j^{e^{-2/3}}\mathbf{C}_j^e$ and $\overline{\mathbf{b}_j^{N,e}} = J_j^{N,e^{-2/3}}\mathbf{b}_j^{N,e}$ where $J_j^e = \det\mathbf{F}_j^e$ and $J_j^{N,e} = \det\mathbf{F}_j^{N,e}$, are the isochoric parts of the total, elastic and network elastic deformation tensors respectively. The parameter κ is the bulk modulus, T_2 is the Kauzmann temperature, λ_L is the limiting stretch in the eight-chain model, μ_j^{neq} and μ_j^{back} are the shear moduli of the P inelastic and network resistant processes respectively. c_{g0} and c_{g1} are the coefficients of the heat capacity of the kinetic subsystem, where $c_g = c_{g0} + c_{g1}T$; Δc_{0_i} and Δc_{1_i} are the coefficients of the heat capacity of the configurational subsystems, where $\Delta c_i = \Delta c_{0_i} + \Delta c_{1_i}T_{e_i}$. We split the inelastic internal energy using a fractional parameter $0 \le a \le 1$, where $\sum_i^Q a_i = a$ is the total configurational contributions to the internal inelastic energy, and $1-a$ is the kinetic contribution. The configurational properties are assumed to be identically distributed over the Q configurational subsystems, therefore $\mu_i{}^{eq} = \phi_i\mu^{eq}, a_i = \phi_i a, \Delta c_{0_i} = \phi_i\Delta c_0, \Delta c_{1_i} = \phi_i\Delta c_1$, where

$0 < \phi_i < 1$ and $\sum_i^Q \phi_i = 1$. The μ^{eq} is the equilibrium (rubbery) shear modulus of the material and Δc_0 and Δc_1 represent the difference between the rubbery and glassy heat capacity, $c_r = (c_{g0} + \Delta c_0) + (c_{g1} + \Delta c_1)T$.

According to eq. (4.24), the second Piola-Kirchhoff stress can be evaluated as $\mathbf{S} = 2\frac{\partial \psi}{\partial \mathbf{C}}$ and the Cauchy stress, $\sigma = \frac{1}{J}\mathbf{F}\mathbf{S}\mathbf{F}^T$ is the push-forward of $\mathbf{S}$,

$$\sigma = \underbrace{\frac{1}{J}(\sum_i^Q \frac{T_{e_i}\phi_i}{T_0})\mu^{eq}(\overline{\mathbf{b}} - \frac{1}{3}\mathrm{tr}(\overline{\mathbf{b}})\mathbf{I})}_{\sigma^{eq}} + \sum_j^P \underbrace{(\frac{1}{J}\mu_j^{neq}(\overline{\mathbf{b}_j^e} - \frac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_j^e})\mathbf{I})}_{\sigma_j^{neq}} + \underbrace{\frac{1}{2J}\kappa(J^2-1)\mathbf{I}}_{p}, \quad (4.40)$$

where $\mathbf{I}$ represents the second-order identity tensor. The stress tensors $\mathbf{M}_j$, $\mathbf{N}_j{}^{back}$ and ζ_j can be evaluated from eq. (4.31) as,

$$\begin{aligned}
&\mathbf{M}_j = \sum_i^Q \phi_i(1 + a(\tfrac{T}{T_{e_i}} - 1))\mu_j^{neq}(\overline{\mathbf{C}}_j^e - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{C}}_j^e)\mathbf{I}),\\
&\mathbf{N}_j^{back} = \mu_j^{back}\frac{\lambda_L}{\bar{\lambda}_{eff,j}}\mathscr{L}^{-1}(\tfrac{\bar{\lambda}_{eff,j}}{\lambda_L})(\overline{\mathbf{b}_j^{N,e}} - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_j^{N,e}})\mathbf{I}),\\
&\zeta_j = \mathbf{M}_j - \mathbf{N}_j^{back}.
\end{aligned} \quad (4.41)$$

From eq. (4.33), the inelastic rate of deformation tensor $\mathbf{D}_j^v$ is related to the flow stress ζ_j through the viscous resistance to local segmental rearrangements ν_j^I while the rate of deformation tensor for network relaxation $\mathbf{D}_j^{N,\nu}$ is related to the backstress $\mathbf{N}_j{}^{back}$ through the viscous resistance to network deformation ν_j^N. ν_j^I and ν_j^N need to be constitutively prescribed. For the viscous resistance to local segmental rearrangements, we adopted the same form as [82], where the Adam-Gibbs model [180] and the Eyring model [174] were

applied to describe the temperature and deformation dependence of viscosity,

$$\mathbf{D}_j^v = \frac{1}{2\nu_j^I}\zeta_j, \nu_j^I = \nu_j^{I,ref}\exp(\frac{B}{T\sum_i^Q \eta_i^c})\frac{V_S^I S}{RT}[\sinh(\frac{V_S^I S}{RT})]^{-1}, \tag{4.42}$$

where B is the thermal activation energy, V_S^I is the activation volume for inelastic deformation and $S = \sqrt{\frac{1}{2}\sum_j^P \zeta_j, : \sum_j^P \zeta_j}$ is the total flow stress. For the viscous resistance to network deformation, we adopted a similar form as [221] to describe its dependence on temperature, strain state and strain rate,

$$\mathbf{D}_j^{N,v} = \frac{1}{2\nu_j^N}\mathbf{N}_j{}^{back}, \nu_j^N = \nu_j^{N,ref}\exp(\frac{A}{T})(\frac{\alpha_0-\alpha_c}{\alpha-\alpha_c})\alpha^{-m}\frac{V_S^N S^{back}}{RT}[\sinh(\frac{V_S^N S^{back}}{RT})]^{-1}, \tag{4.43}$$

where A is a temperature parameter that describes the temperature dependence of network relaxation, V_S^N is the activation volume for network relaxation and $S^{back} = \sqrt{\frac{1}{2}\sum_j^P \mathbf{N}_j^{back} : \sum_j^P \mathbf{N}_j^{back}}$ is the total backstress. α is a chain orientation parameter as defined by Dupaix and Boyce [221], $\alpha = \pi/2 - \max\{\alpha_1, \alpha_2, \alpha_3\}$, where $(\alpha_1, \alpha_2, \alpha_3)$ are the angles between the end-to-end vector of a polymer chain and each of the three principal axes. The polymer chains rotate and stretch to align in the direction of deformation and α describes the extent of chain alignment with a principle axis. The parameter m is a power-law exponent for α and α_0 is the initial value of α, which is 0.616 rad considering a unit cube in the eight-chain model. The parameter α_c is the orientation level where the relaxation cutoff occurs, which was introduced to avoid underestimating strain hardening at very large strains [221].

In the following sections, we neglected the term of effective temperature diffusion in eq. (4.37). For the free energy density shown in eq. (4.38), the evolution equation for effective temperatures eq. (4.37) reduces to,

$$\dot{T_{e_i}} = \frac{T-T_{e_i}}{\tau_{R_i}} + \frac{a}{\Delta c_0+\Delta c_1 T_{e_i}} \sum_j^P \mu_j^{neq} (\overline{\mathbf{C}}_j^e - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{C}}_j^e)\mathbf{I}) : \mathbf{D}_j^v + \frac{J^{-2/3} T_{e_i} \mu^{eq}}{2(\Delta c_0+\Delta c_1 T_{e_i}) T_0} (\mathbf{I} - \tfrac{1}{3}\mathrm{tr}(\mathbf{C})\mathbf{C}^{-1}) : \dot{\mathbf{C}},$$

$$\tau_{R_i} = {\tau_{R_i}}^{ref} \exp(\frac{B}{T \sum_i^Q \eta_i^c}),$$

(4.44)

where we have applied the Adam-Gibbs model [180] for the structural relaxation times and we assumed that the inelastic viscosities ν_j^I and the structural relaxation times τ_{R_i} share the same temperature and nonequilibrium structural dependence [82]. The entropy dependence in the Adam-Gibbs model is the key to capture the effects of physical aging and mechanical rejuvenation. The physical aging reduces the configurational entropy and therefore increases the deformation resistance, while the mechanical rejuvenation reverses this effect by increasing the configurational entropy.

4.2.3 Finite element simulation methods

The above constitutive model was implemented into an open source finite element package TAHOE (Sandia National Laboratories). To acquire the structural relaxation spectrum, we applied the model to simulate the DSC experiments. The temperature field and the displacement field were solved from the thermomechanical boundary value problems using a staggered scheme as described in [234].

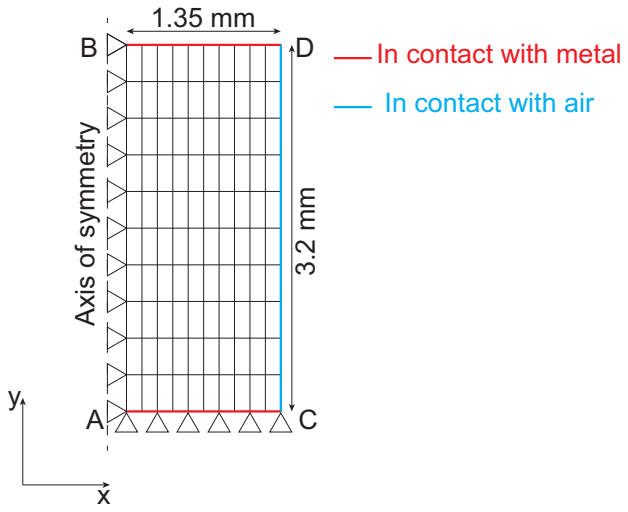


Figure 4.2: Finite element model of an axisymmetric cylindrical specimen.

Fig. 4.2 shows a schematic of an axisymmetric model, where the geometry was discretized using bilinear quadrilateral elements. The displacement boundary condition was $u_x(AB) = u_y(AC) = 0$. The remaining surfaces were assumed to be traction free. Convection boundary conditions were applied on surfaces BD, DC and AC for the thermal diffusion problem. Surfaces AC and BD were assumed to be in contact with the DSC aluminum pan and lid, and the heat transfer between the specimen and metal can be described by $\mathbf{q} = h_1(T - T_{DSC})\mathbf{n}$, where h_1 is the heat transfer coefficient between polymer and metal, $\mathbf{n}$ is the normal direction of the surface and T_{DSC} is the set temperature in the DSC experiments. Surface CD was assumed to be in contact with air, and the heat transfer between the specimen and air was described by $\mathbf{q} = h_2(T - T_{DSC})\mathbf{n}$, where h_2 is the heat transfer coefficient between polymer and air. In the following simulations, we adopted $h_1 = 7500\mathrm{W}/(\mathrm{m}^2\mathrm{K})$ and $h_2 = 10\mathrm{W}/(\mathrm{m}^2\mathrm{K})$ [234, 235]. The macroscopic specific heat flux

can be calculated from eqs. (4.35) and (4.36) as,

$$\begin{aligned}\overline{Q} = \tfrac{1}{\rho V}\int \Bigg(& (c_{g0}+c_{g1}T)\,\dot{T} + \sum_{i}^{Q}\phi_i(\Delta c_0 + \Delta c_1 T_{e_i})\dot{T}_{e_i} \\ & - \sum_{j}^{P}\mu_j^{neq}(\overline{\mathbf{C}}_j^e - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{C}}_j^e)\mathbf{I}) : \mathbf{D}_j^v - J\sigma^{eq} : \mathbf{D} + \sum_{j}^{P}\mathbf{N}_j^{back} : \mathbf{D}_j^{N,e} \Bigg)\, dV,\end{aligned} \tag{4.45}$$

where $\mathbf{D} = \frac{1}{2}(\dot{\mathbf{F}}\mathbf{F}^{-1} + \mathbf{F}^{-T}\dot{\mathbf{F}}^{T})$ is the total rate of deformation tensor, $\rho = 1.2\mathrm{g/cm}^3$ is the density of polymer and V is the volume of the specimen.

4.3 Model Parameters

Three relaxation spectra describe the time-dependent and rate-dependent behavior of the material, a structural relaxation spectrum (τ_{R_i}, ϕ_i), a stress relaxation spectrum $(\tau_{S_j}, \mu_j^{neq})$ for local segmental motion and a stress relaxation spectrum for the long range network deformation $(\tau_{N_j}, \mu_j^{back})$. Here we express the relaxation times using T_g as the reference temperature for the purpose of parameter determination. The configurational entropy can be evaluated from eq. (4.24) as,

$$\begin{aligned}\eta^c(T_{e_i}, \mathbf{C}) &= \sum_{i}^{Q}\eta_i^c \\ &= \sum_{i}^{Q}\Big(\Delta c_0\phi_i \log\tfrac{T_{e_i}}{T_2} + \Delta c_1\phi_i(T_{e_i}-T_2) - \tfrac{\phi_i}{T_0}\tfrac{\mu^{eq}}{2}(\mathrm{tr}(\overline{\mathbf{C}})-3)\Big).\end{aligned} \tag{4.46}$$

The stress relaxation times are defined as $\tau_{S_j} = \nu_j^I/\mu_j^{neq}$ and $\tau_{N_j} = \nu_j^N/\mu_j^{back}$. Thus, from eqs. (4.42), (4.43) and (4.44), the relaxation times at $\mathbf{F} = \mathbf{I}$ and $T = T_{e_i} = T_g$ can be evalu-

ated as,

$$\begin{aligned}
\tau^g_{R_i} &= \tau^{ref}_{R_i} \exp\left(\frac{B}{T_g \eta^c(T_g,\mathbf{I})}\right), i = 1...Q, \\
\tau^g_{S_j} &= \tau^{ref}_{S_j} \exp\left(\frac{B}{T_g \eta^c(T_g,\mathbf{I})}\right), j = 1...P, \\
\tau^g_{N_j} &= \tau^{ref}_{N_j} \exp\left(\frac{A}{T_g}\right) \alpha_0^{-m}, j = 1...P.
\end{aligned} \tag{4.47}$$

Substituting above equations to eqs. (4.42), (4.43) and (4.44) yields relaxation times at general state,

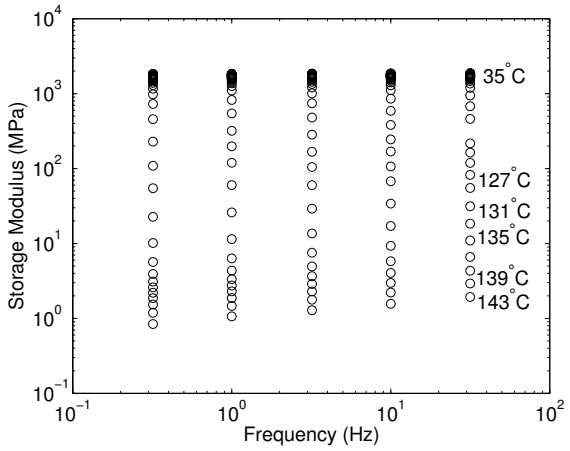
$$\begin{aligned}
\tau_{R_i} &= {\tau_{R_i}}^g \exp\left(\frac{B}{T\eta^c(T_{e_i},\mathbf{C})} - \frac{B}{T_g\eta^c(T_g,\mathbf{I})}\right), \\
\tau_{S_j} &= {\tau_{S_j}}^g \exp\left(\frac{B}{T\eta^c(T_{e_i},\mathbf{C})} - \frac{B}{T_g\eta^c(T_g,\mathbf{I})}\right) \frac{V^I_S S}{RT} [\sinh(\frac{V^I_S S}{RT})]^{-1}, \\
\tau_{N_j} &= {\tau_{N_j}}^g \exp\left(\frac{A}{T} - \frac{A}{T_g}\right)\left(\frac{\alpha_0-\alpha_c}{\alpha-\alpha_c}\right)\left(\frac{\alpha}{\alpha_0}\right)^{-m} \frac{V^N_S S_{back}}{RT} [\sinh(\frac{V^N_S S_{back}}{RT})]^{-1}.
\end{aligned} \tag{4.48}$$

The details of the parameter determination process are discussed in the following sections. Table. 4.1 lists the parameters, their physical significance and values determined for polycarbonate.

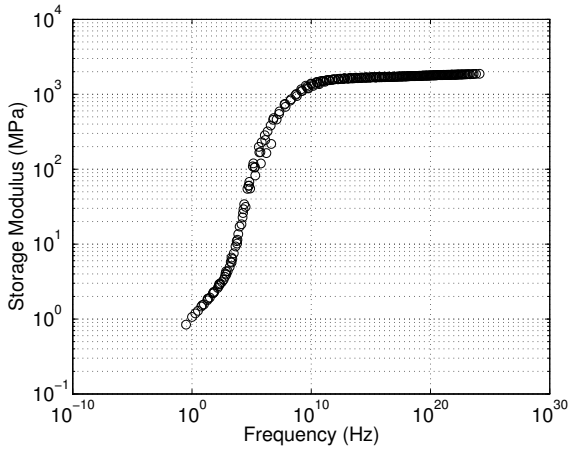
4.3.1 Determining the stress relaxation spectrum

The master curve of storage modulus was applied to determine the stress relaxation spectrum. The storage modulus measured at different temperatures was plotted on a log-log scale in Fig. 4.3(a) and it was shifted horizontally to a reference temperature $T_0 = 143°\text{C}$ to form the master curve for the frequency-dependence of the storage modulus as shown in Fig. 4.3(b). The master curve does not have a plateau at low frequency (or high temperature) because of its thermoplastic nature, therefore the equilibrium modulus μ^{eq} was

set to be zero. The temperature dependence of the shift factor in the glass transition region can be fitted by the WLF empirical relation as described in Sec. 4.2.1.2. Fitting the WLF relation to the shift factors gave $C_1^0 = 9.7136$ and $C_2^0 = 65.572$K. The onset glass transition temperature $T_g = 110^\circ$C was determined from the starting temperature that the shift factors can be predicted by the WLF relation, as shown in Fig. 4.4. The nonequilibrium Young's modulus $E^{neq} = 1876.8$MPa was approximated by the maximum storage modulus of the master curve in Fig. 4.3(b) and the nonequilibrium shear modulus $\mu^{neq} = 695.1$MPa was obtained from the relation $\mu^{neq} = E^{neq}/2(1+\nu_g)$ assuming a glassy Poisson's ratio $\nu_g = 0.35$. The bulk modulus can be evaluated as $\kappa = E^{neq}/(3(1-2\nu_g))$.



(a)



(b)

Figure 4.3: Storage modulus as a function of frequency: (a) measured for different temperature and (b) shifted to a reference temperature of $T_0 = 143^{\circ}$C to form a master curve.

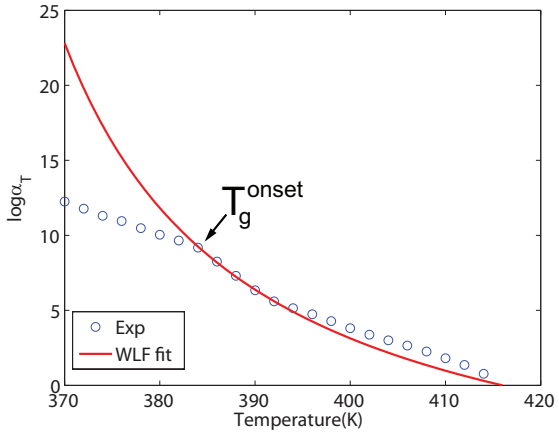


Figure 4.4: Shift factor as a function of temperature and the WLF fit

We developed a second-order approximation method based on Schwarzl and Staverman [121] to determine the discrete relaxation spectrum [83]. We first used a polynomial function denoted as $\log G' = f_0(\log \omega)$ to fit the master curve in Fig. 4.3(b), where G' is the storage modulus and ω represents the angular frequency. The relation between relaxation modulus and a continuous relaxation spectrum was defined as [231],

$$G(t) = \mu^{eq} + \int_0^\infty h(\nu) e^{-\nu t} d\nu, \tag{4.49}$$

where $h(\nu)$ is the continuous relaxation spectrum. The cumulative relaxation spectrum is defined from $h(\nu)$ as,

$$H(\nu) = \int_0^\nu h(z) dz. \tag{4.50}$$

Evaluating $h(\nu)$ requires inverting the integral equation (4.49) which can be challenging. A number of approximations have been developed. In particular, we used a second-order

accurate approximation developed by Schwarzl and Staverman [121],

$$h(\nu) = \frac{\sqrt{2}}{\omega} G'(\omega)[d\log G'/d\log\omega - \tfrac{1}{2}(d\log G'/d\log\omega)^2 - \tfrac{1}{4.606} d^2\log G'/d(\log\omega)^2]|_{\omega=\sqrt{2}\nu}. \quad (4.51)$$

Applying the polynomial fit to eqs. (4.51) and (4.50), we can calculate the continuous relaxation spectrum and cumulative distribution from polynomial function f_0 as,

$$h(\nu) = 10^{f_0(x)-x}\sqrt{2}(f_0{}'(x) - \frac{1}{2}(f_0{}'(x))^2 - \frac{1}{4.606} f_0{}''(x))|_{x=\log\sqrt{2}\nu}, \quad (4.52)$$

$$H(\nu) = 10^{f_0(x)}(1 - \frac{1}{2} f_0{}'(x))|_{x=\log\sqrt{2}\nu}. \quad (4.53)$$

The relaxation modulus of the discrete model can be written as,

$$G_{disc}(t) = \mu^{eq} + \sum_j^P \mu_j^{neq} \exp(-t\nu_j). \quad (4.54)$$

The discrete cumulative spectrum can be evaluated by combining eqs. (4.49), (4.50) and (4.54) as,

$$H_{disc}(\nu) = \sum_j^P \mu_j^{neq} \left\langle \nu - \nu_j \right\rangle^1 - \mu^{eq}, \quad (4.55)$$

where $\langle\rangle^1$ denotes a step function. We then assume a power law distribution for the relaxation frequencies,

$$\nu_j^0 = \nu_{\min}^0 \left(\frac{\nu_{\max}^0}{\nu_{\min}^0} \right)^{(j-1)/(P-1)}, \quad (4.56)$$

where $\nu^0_{\max}$ and $\nu^0_{\min}$ are the maximum and minimum of relaxation frequencies and are selected based on the frequency range of the master curve. Finally, the nonequilibrium moduli μ_j^{neq} corresponding to the relaxation frequencies ν_j^0 were determined from the continuous cumulative distribution as follows [229, 236],

$$\begin{aligned}
\mu_1^{neq} &= \tfrac{1}{2}(H(\nu_1^0) + H(\nu_2^0)), \\
\mu_j^{neq} &= \tfrac{1}{2}(H(\nu_{j+1}^0) - H(\nu_{j-1}^0)), 1 < j < P-1, \\
\mu_P^{neq} &= \mu^{neq} - \sum_j^{P-1} \mu_j^{neq}.
\end{aligned} \tag{4.57}$$

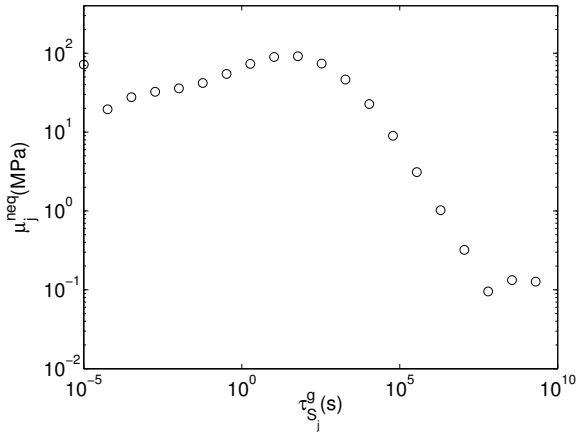
The stress relaxation time ${\tau^g}_{S_j}$ can be evaluated from ν_j^0,

$${\tau^g}_{S_j} = \frac{1}{\nu_j^0} \exp\left(\frac{B}{T_g \eta^c(T_g, \mathbf{I})} - \frac{B}{T_0 \eta^c(T_0, \mathbf{I})} \right). \tag{4.58}$$

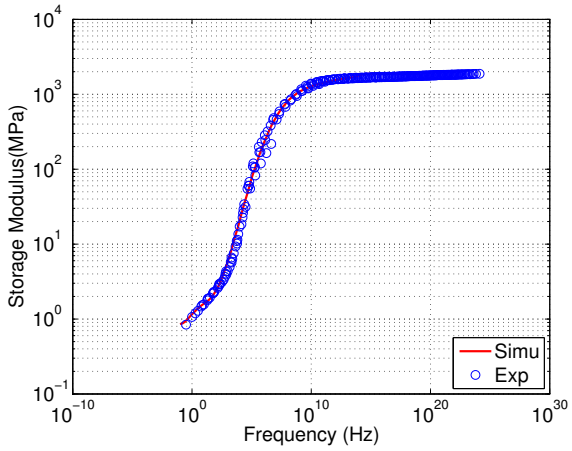
Fig. 4.5(a) plots the stress relaxation spectrum $(\tau^g_{S_j}, \mu_j^{neq})$ with $P = 20$ processes determined as described above. The spectrum was applied to evaluate the storage modulus of the discrete model,

$$E'(\omega) = E^{eq} + \sum_j^P \frac{E_j^{neq} \omega^2 / (\nu_j^0)^2}{1 + \omega^2 / (\nu_j^0)^2}. \tag{4.59}$$

The results shown in Fig. 4.5(b) show excellent agreement with the measured master curve.



(a)



(b)

Figure 4.5: (a) Distribution of the discrete stress relaxation spectrum $(\tau_{S_j}^g, \mu_j^{neq})$. (b) Comparison between master curves from experiments and discrete model.

4.3.2 Determining the structural relaxation spectrum and heat capacities

To determine the structural relaxation spectrum, we assumed that the structural relaxation spectrum can be described by the Kohlrausch-Williams-Watts (KWW) model [82, 230]. The continuous relaxation spectrum for the KWW model can be expressed as infinite series [237],

$$h_{KWW}(\tau) = -\frac{\chi}{\pi\tau^2}\sum_{k=0}^{\infty}\frac{(-1)^k}{k!}\sin(\pi\beta k)\Gamma(\beta k+1)\left(\frac{\tau}{\chi}\right)^{\beta k+1}, \tag{4.60}$$

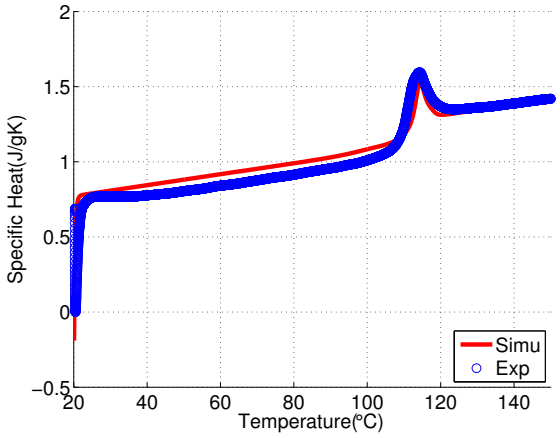
where χ is the characteristic structural relaxation time, $0 \leq \beta \leq 1$ represents the breadth of the structural relaxation spectrum and $\Gamma()$ is the gamma function. We also assumed a power law distribution for discrete structural relaxation times,

$$\tau^g_{R_i} = \tau_{\min}\left(\frac{\tau_{\max}}{\tau_{\min}}\right)^{(i-1)/(Q-1)}. \tag{4.61}$$

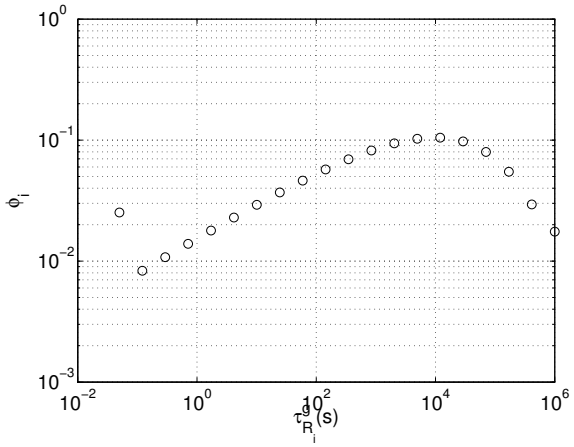
The discrete structural relaxation spectrum were determined by a stepwise approximation of the cumulative relaxation spectrum $H_{KWW}(\tau) = \int_0^\tau h_{KWW}(t)dt$ as,

$$\begin{aligned}
&\phi_1 = \tfrac{1}{2}(H_{KWW}(\tau^g{}_{R_1}) + H_{KWW}(\tau^g{}_{R_2})),\\
&\phi_i = \tfrac{1}{2}(H_{KWW}(\tau^g{}_{R_{i+1}}) - H_{KWW}(\tau^g{}_{R_{i-1}})), 1 < i < Q-1,\\
&\phi_Q = 1 - \sum_i^{Q-1}\phi_i.
\end{aligned} \tag{4.62}$$

We simulated the DSC experiments using the method in Sec. 4.2.3 and tuned the characteristic time $\chi = 5000$s and breadth $\beta = 0.30$ to fit the position, width and height of the DSC endothermic overshoot (Fig. 4.6(a)). The structural relaxation spectrum with $Q = 20$ processes was plotted in Fig. 4.6(b).



(a)



(b)

Figure 4.6: (a) Comparison between the measured and model predicted DSC response. (b) Distribution of the discrete structural relaxation spectrum (τ_{R_i}, ϕ_i).

The coefficients for heat capacities can be obtained through linear fits of the DSC curve in different temperature ranges. Specifically, the kinetic components c_{g0} and c_{g1} were determined from a linear fit of the DSC curve from 40°C to 60°C, while the heat capacity for the whole system was determined from a linear fit of the DSC curve from 130°C to 150°C. Subtracting the kinetic components from the heat capacity of the whole system gives the configurational components of heat capacity Δc_0 and Δc_1.

4.3.3 Determining the temperature-dependent and stress-dependent parameters

The activation volume for inelastic deformation V_S^I was obtained through fitting the yield stress in uniaxial compression at 37°C and three different strain rates 10^{-3}/s, 10^{-4}/s, 10^{-5}/s. The viscous resistance to local segmental rearrangements ν^I at the yield point was estimated as yield stress divided by strain rate. Therefore we can plot $1/\nu^I$ against the yield stress and use the Eyring model to fit the experiment data, as shown in Fig. 4.7, from which V_S^I was determined. The parameters B and T_2 were determined by fitting the Adam-Gibbs model $\alpha(T) = \exp(\frac{B}{T\eta^c(T,\mathbf{I})} - \frac{B}{T_0\eta^c(T_0,\mathbf{I})})$ to the experiment measured shift factors in the glass transition region, as shown in Fig. 4.8. The fraction parameter $0 < a < 1$ controls the rate of strain softening and was determined through fitting to the stress response of specimens deformed at 37°C and strain rate of 10^{-4}/s.

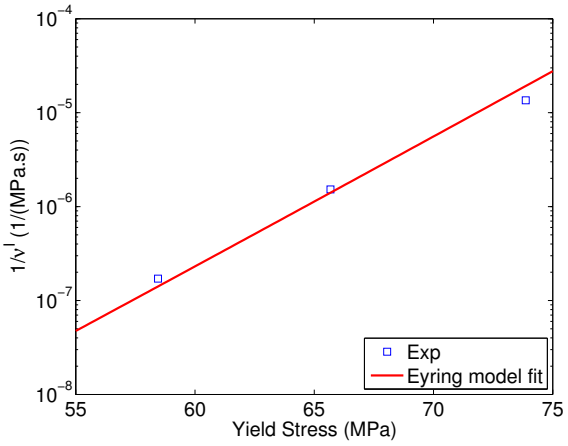


Figure 4.7: Inverse of viscosity versus yield stress, comparing the experiment data at 37°C and three different strain rates 10^{-3}/s, 10^{-4}/s, 10^{-5}/s and the Eyring model fit.

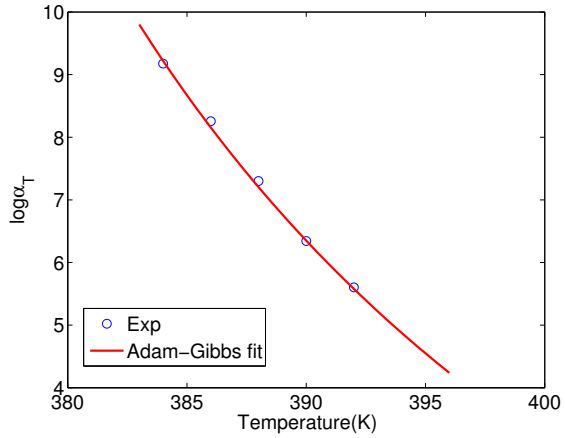


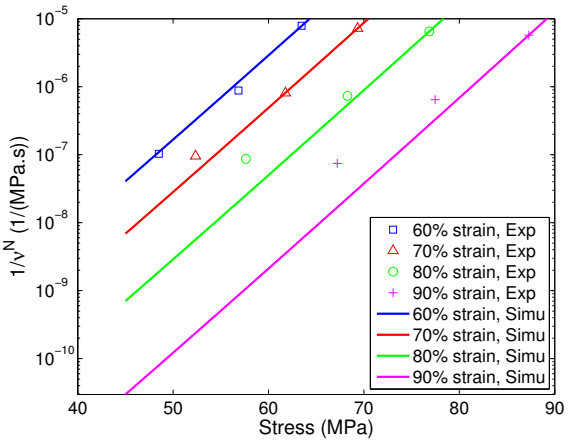
Figure 4.8: Shift factors at the glass transition region and the Adam-Gibbs model fit

The activation volume for network relaxation V_S^N was determined in a similar way with V_S^I. Assuming that the network resistance is the dominant mechanism at large strain, we picked the stress at 60%, 70%, 80% and 90% true strain for specimens deformed at 37°C and strain rates of 10^{-3}/s, 10^{-4}/s, 10^{-5}/s and estimated the viscous resistance to network deformation as stress divided by strain rates. The activation volume V_S^N can be obtained

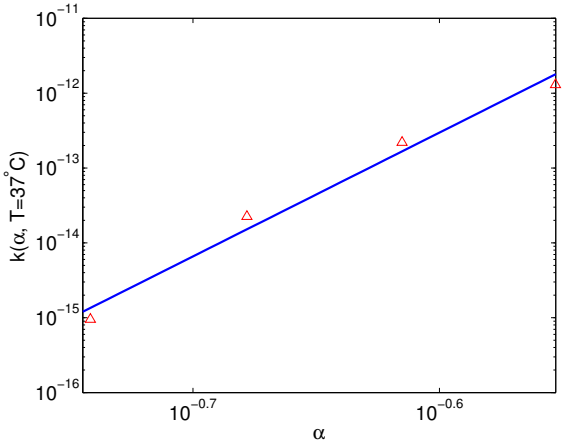
through fitting the Eyring model to the experiment data in a $1/\nu^N$ vs. stress plot, as shown in Fig. 4.9(a). From eq. (4.43), we can express the viscous resistance to network deformation as,

$$\frac{1}{\nu^N} = \frac{1}{\nu^{N,ref}} k(\alpha, T) \frac{RT}{V_S^N S^{back}} [\sinh(\frac{V_S^N S^{back}}{RT})], \tag{4.63}$$

where $k(\alpha, T)$ is a function that depends on chain orientation and temperature and its value for $T = 37^\circ$C can be obtained from the curve fittings in Fig. 4.9(a). Therefore we can plot $k(\alpha, 37^\circ\text{C})$ against α at four different strain levels in log-log scale and the slope of a linear fit gives the power-law exponent m, as shown in Fig. 4.9(b). For the cessation parameter α_c, we adopted the same value 0.05 as [221]. Repeating this process for $T = 95^\circ$C and comparing the values of $k(\alpha, T)$ at same α (strain level) but different temperature gives the value of the temperature parameter $A = 6949.5$K.



(a)



(b)

Figure 4.9: (a) Inverse of viscosity versus stress at four strain levels (60%, 70%, 80% and 90%), for specimens deformed at 37° and strain rates of 10^{-3}/s, 10^{-4}/s, 10^{-5}/s, comparing the experiment data and the Eyring model fit. (b) Function $k(\alpha, T)$ versus α at 37°C.

4.3.4 Determining the stress relaxation spectrum for network deformation and the limiting stretch

We assumed that the network relaxation spectrum can be obtained through vertical and horizontal shifting of the stress relaxation spectrum, i.e. $(\tau^g_{N_j}, \mu^{back}_j) = (t_x\tau^g_{S_j}, t_y\mu^{neq}_j)$. The shift factors t_x, t_y and the limiting stretch λ_L were determined through fitting the stress response of specimens deformed at 37°C and 10^{-4}/s. Specifically, the shift factors t_x, t_y were determined by fitting the measured hardening modulus while the limiting stretch λ_L was determined from the transition strain level between Gaussian hardening and Langevin hardening. The discrete network relaxation spectrum with $P = 20$ processes is shown in Fig. 4.10.

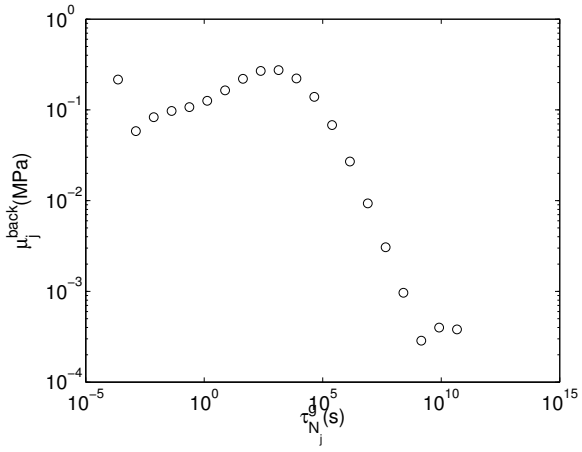


Figure 4.10: Discrete network relaxation spectrum $(\tau^g_{N_j}, \mu^{back}_j)$, obtained through shifting of the stress relaxation spectrum $(\tau^g_{S_j}, \mu^{neq}_j)$.

Parameter	Physical significance	Values
T_g(K)	Glass transition temperature	383

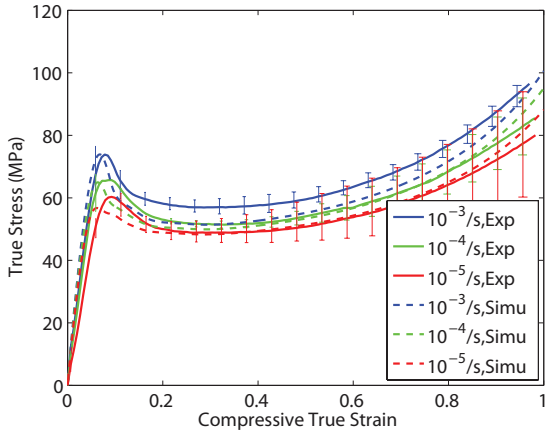
μ^{eq} (MPa)	Equilibrium shear modulus	0
κ (MPa)	Bulk modulus	2085.3
$V_S^I(\text{cm}^3/\text{mol})$	Activation volume for viscous flow	860.63
B(J/g)	Thermal activation energy	357.0
T_2(K)	Kauzmann temperature	350.428
T_0(K)	Reference temperature	416
c_{g0}(J/(gK))	Coefficient of heat capacity of kinetic subsystem	-0.3005
c_{g1}(J/(gK))	Coefficient of heat capacity of kinetic subsystem	0.003665
Δc_0(J/(gK))	Coefficient of excess heat capacity of configurational subsystems	0.2822
Δc_1(J/(gK))	Coefficient of excess heat capacity of configurational subsystems	-0.0001112
a	Configurational contribution to the internal inelastic energy	0.5
$(\tau_{S_j}^g, \mu_j^{neq})$	Discrete stress relaxation spectrum	-
$(\tau_{R_i}^g, \phi_i)$	Discrete structural relaxation spectrum	-
$(\tau_{N_j}^g, \mu_j^{back})$	Discrete network relaxation spectrum	-
A(K)	Temperature coefficient	6949.5
$V_S^N(\text{cm}^3/\text{mol})$	Activation volume for network relaxation	782.2
m	Power of the orientation parameter	16.5418
α_c	Cessation value of the orientation parameter for network relaxation	0.05
α_0	Initial value of the orientation parameter	0.616
λ_L	Limiting stretch	2.80

Table 4.1: Parameters of the constitutive model for polycarbonate.

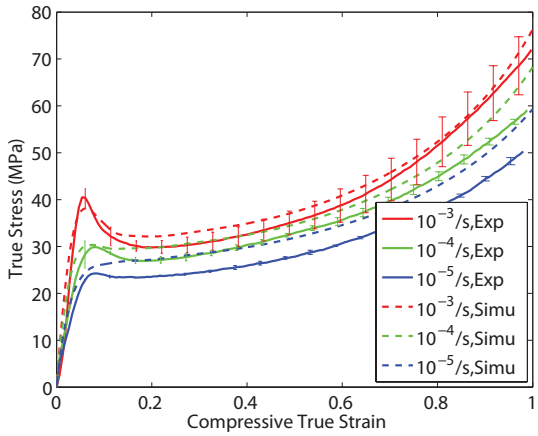
4.4 Results

Eqs. (4.40)-(4.44) summarized the constitutive model and the parameters for polycarbonate were determined as described in Sec. 4.3. We applied the constitutive model to simulate the stress response in uniaxial compression experiments at different temperature and strain rates, as described in Sec. 4.2.1.4. The thermal history in simulations was exactly the same with that in experiments. Only the experimental results at 37°C and 10^{-4}/s were used to determine the model parameters as described in Sec 4.3.4. The remaining experimental results were outside the range of model calibration.

Fig. 4.11 plots the uniaxial compression stress response at different true strain rates 10^{-3}/s, 10^{-4}/s and 10^{-5}/s for 37°C and 95°C, comparing the experimental results and the model predictions. The experiment data shown in the plots were obtained through taking average over multiple specimens and the error-bars denote the variation between these specimens. The model was able to capture the major features of the stress response at different deformation level, including the yielding peak, the post-yielding softening and the strain hardening up to 100% true strain.



(a)

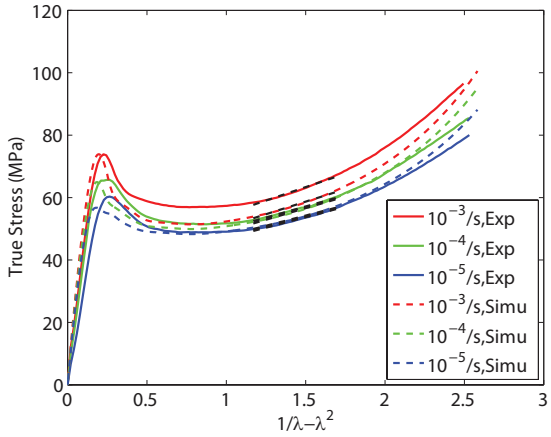


(b)

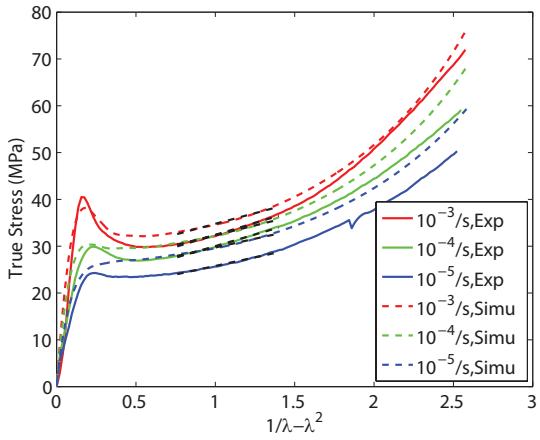
Figure 4.11: Uniaxial compression response of polycarbonate at different true strain rates 10^{-3}/s, 10^{-4}/s and 10^{-5}/s. (a)37°C and (b) 95°C.

In order to quantitatively investigate the strain hardening behavior, we plotted the stress against the negative Gaussian hardening function $g(\lambda) = -(\lambda^2 - 1/\lambda)$ as defined in eq. (4.1). As shown in Fig. 4.12, the material initially exhibits Gaussian hardening, where the stress is linear with $g(\lambda)$, while the behavior becomes highly nonlinear at large strain. This

transition was well captured by the model predictions. We applied linear fits to the Gaussian hardening region to determine the hardening modulus G_R, as indicated by the black dash lines in Fig. 4.12 [191]. Fig. 4.13 plots the hardening modulus at different strain rates for 37°C and 95°C, from which the rate dependence of hardening can be concluded. In both experiments and simulation, the hardening modulus increases with strain rate in the range from 10^{-5}/s to 10^{-3}/s. The model was able to accurately capture this rate dependence, with the difference in hardening modulus less than 2 MPa between experiment results and model predictions.

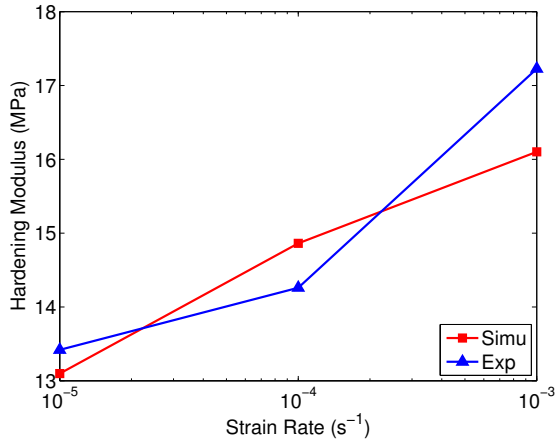


(a)

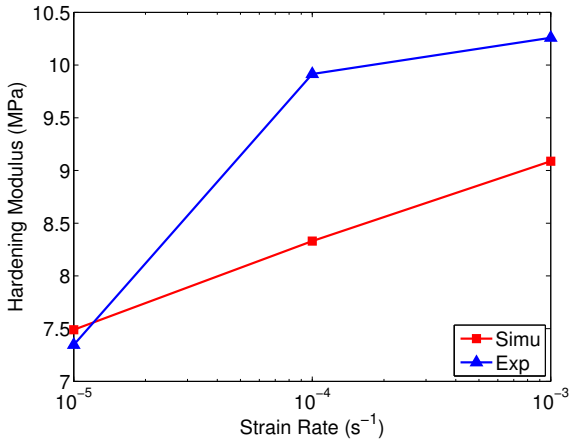


(b)

Figure 4.12: True stress versus the Gaussian hardening function for polycarbonate specimens deformed at different true strain rates 10^{-3}/s, 10^{-4}/s and 10^{-5}/s. (a)37°C and (b) 95°C. Dashed black lines indicate linear fits used to determine hardening modulus.



(a)



(b)

Figure 4.13: Hardening modulus for polycarbonate specimens deformed at different true strain rates 10^{-3}/s, 10^{-4}/s and 10^{-5}/s. (a)37°C and (b) 95°C.

To investigate the temperature dependence of strain hardening, the experiment and simulation results in Fig. 4.11 were grouped by strain rates instead of temperature, as shown in Fig. 4.14, and the hardening modulus at different temperatures were plotted in Fig. 4.15. Compared to the rate dependence in Fig. 4.13, the hardening modulus exhibited stronger

temperature dependence. In both experiments and simulation, the hardening modulus decrease approximately linearly with temperature in the range from 15°C to 95°C. The results shown in Fig. 4.14 and Fig. 4.15 again demonstrated good agreement between experiments and simulation.

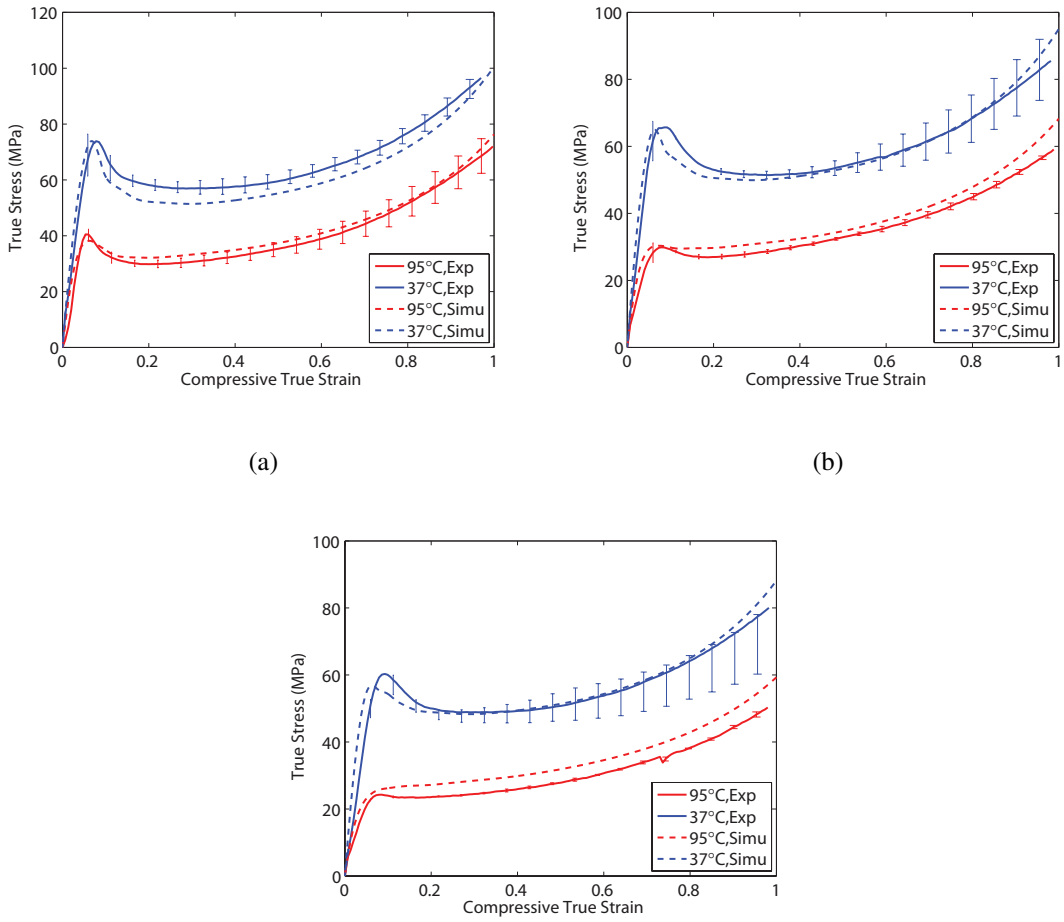


Figure 4.14: Uniaxial compression response of polycarbonate at 37°C and 95°C. (a)10^{-3}/s (b) 10^{-4}/s and (c) 10^{-5}/s.

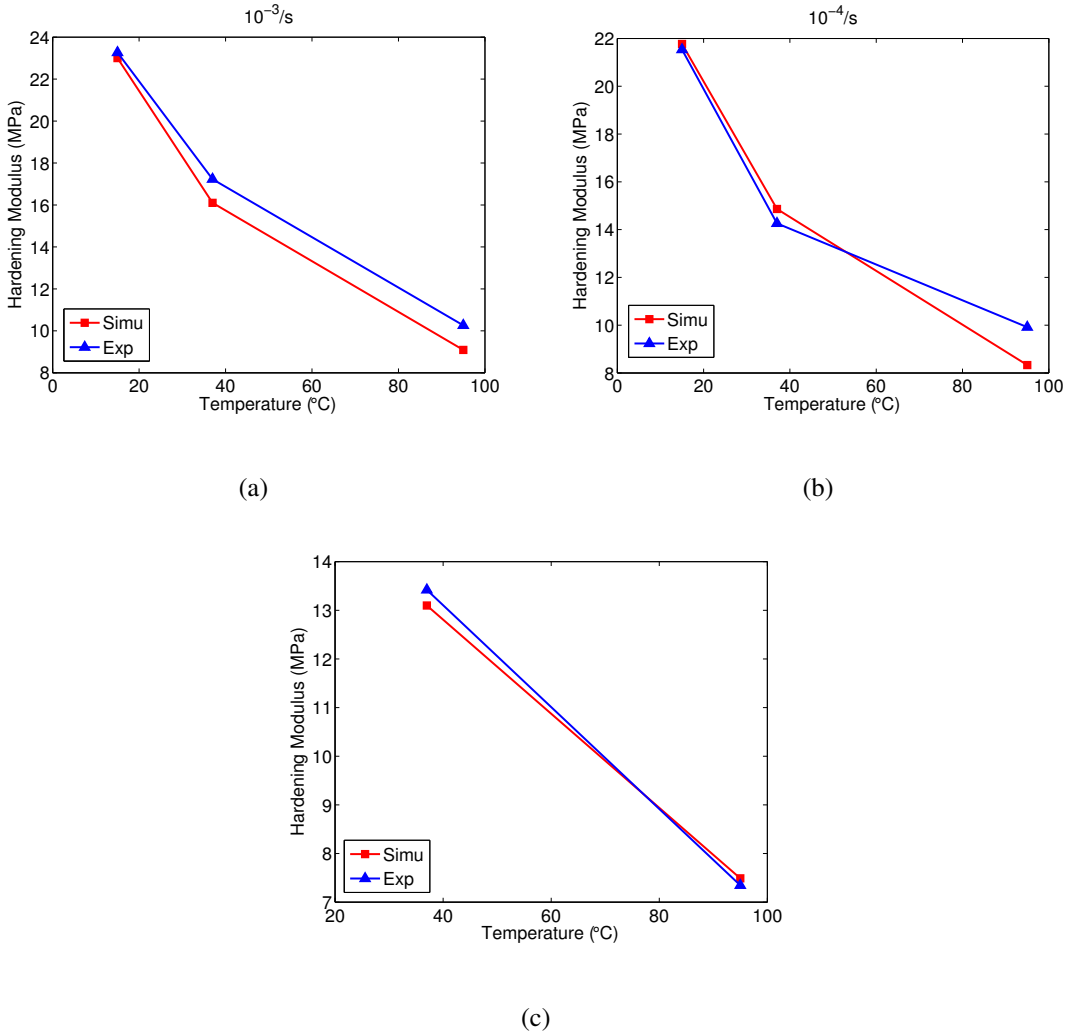
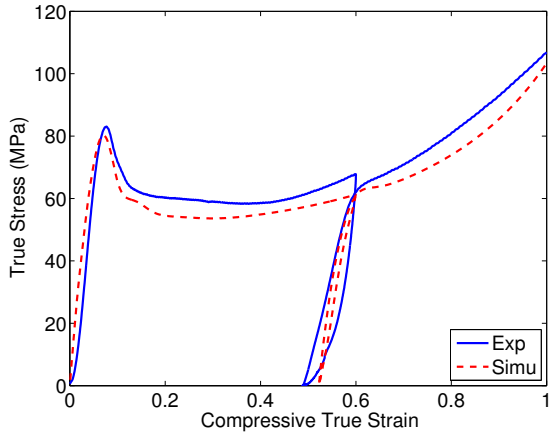


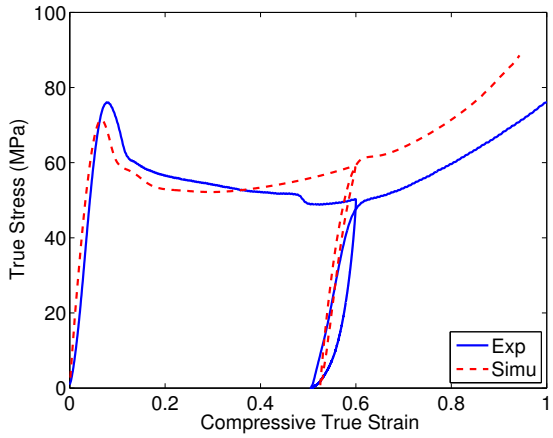
Figure 4.15: Hardening modulus for polycarbonate specimens deformed at different temperatures 15°C, 37°C and 95°C. (a)10^{-3}/s, (b)10^{-4}/s and (c) 10^{-5}/s.

We also simulated the loading-unloading-reloading response of the specimens at room temperature and two strain rates10^{-3}/s and 10^{-4}/s. As shown in Fig. 4.16, in both experiments and simulation, the strain softening was eliminated in the reloading phase because of the effects of mechanical rejuvenation, and the pre-deformed specimens follow the path of specimens without pre-deformation in terms of strain hardening, as discussed in [17]. We

were able to get some hysteresis between unloading and reloading in the simulation, but the degree of it was underestimated when compared to experiments as shown in Fig. 4.16.



(a)

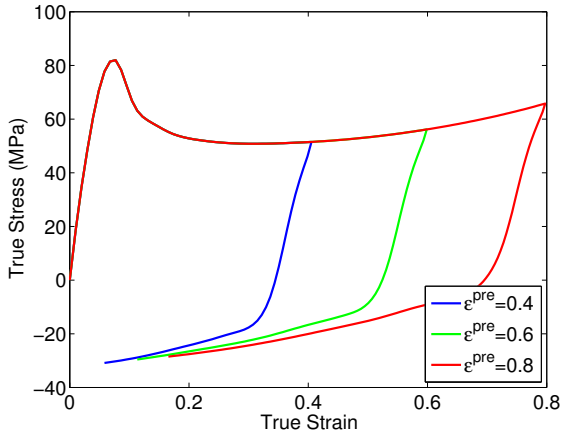


(b)

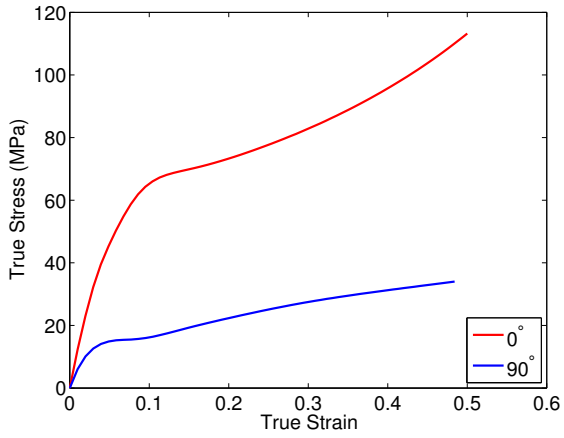
Figure 4.16: Loading-unloading-reloading response of polycarbonate specimens deformed at room temperature (27°C) and engineering strain rates of (a)10^{-3}/s and (b) 10^{-4}/s.

The model was also applied to qualitatively simulate experiments reported in literatures and it was able to capture phenomena related to strain hardening, such as the Bauschinger

effect [17, 18] and the deformation induced anisotropy [19, 238]. In these simulations, the parameters measured for PC in Sec. 4.3 were applied. In Fig. 4.17(a), the specimens were stretched to 40%, 60% and 80% true strain at 0.01/s and 37°C, unloaded to zero force and compressed at the same strain rate and temperature. The model was able to predict the Bauschinger effect, that is, the yield stress in compression is smaller in magnitude than that in tension. In Fig. 4.17(b), the specimens were stretched to 80% true strain along one axis at 0.01/s and 37°C, unloaded to zero force and further stretched along the original axis or 90° to the original axis at the same strain rate and temperature. The model was capable of describing the deformation induced anisotropy, that is, the yield stress and the hardening modulus are higher when deformed parallel to the pre-deformation, as a result of chain orientation.



(a)



(b)

Figure 4.17: Qualitative simulations of experiments reported in literature (a) simulation of the Bauschinger effect [17, 18] and (b) simulation of the deformation induced anisotropy [19].

4.5 Parameter Study

In this section, we investigated the effects of some key parameters in the model that control the temperature dependence, strain rate dependence and strain dependence of hardening. These dependences were incorporated in the constitutive model through the viscous resistance to network deformation, as expressed in eq. (4.43). The activation temperature A controls the temperature dependence of hardening. To examine its effects, we ran simulations with $A = 6949.5$K and $A = 3000$K at three different temperatures and compared the temperature dependence of the hardening modulus. As shown in Fig. 4.18(a), a larger value of A results in a larger hardening modulus as well as a stronger temperature dependence. On the other hand, the strain rate dependence of hardening is affected by the activation volume for network relaxation V_S^N. This is demonstrated in Fig. 4.18(b), where we compared the rate dependence of hardening modulus for two different values of V_S^N. A smaller activation volume leads to a larger hardening modulus and stronger strain rate dependence. The parameter m, which is the power of the orientation parameter, describes the degree in which molecular orientation increases the resistance to network relaxation, hence has effects on the strain-dependence of hardening. These effects are shown in Fig. 4.18(c), where we plotted the stress response at same temperature 95°C and strain rate 10^{-3}/s, but with three different values of m. A higher m produces a stiffer and more nonlinear hardening response.

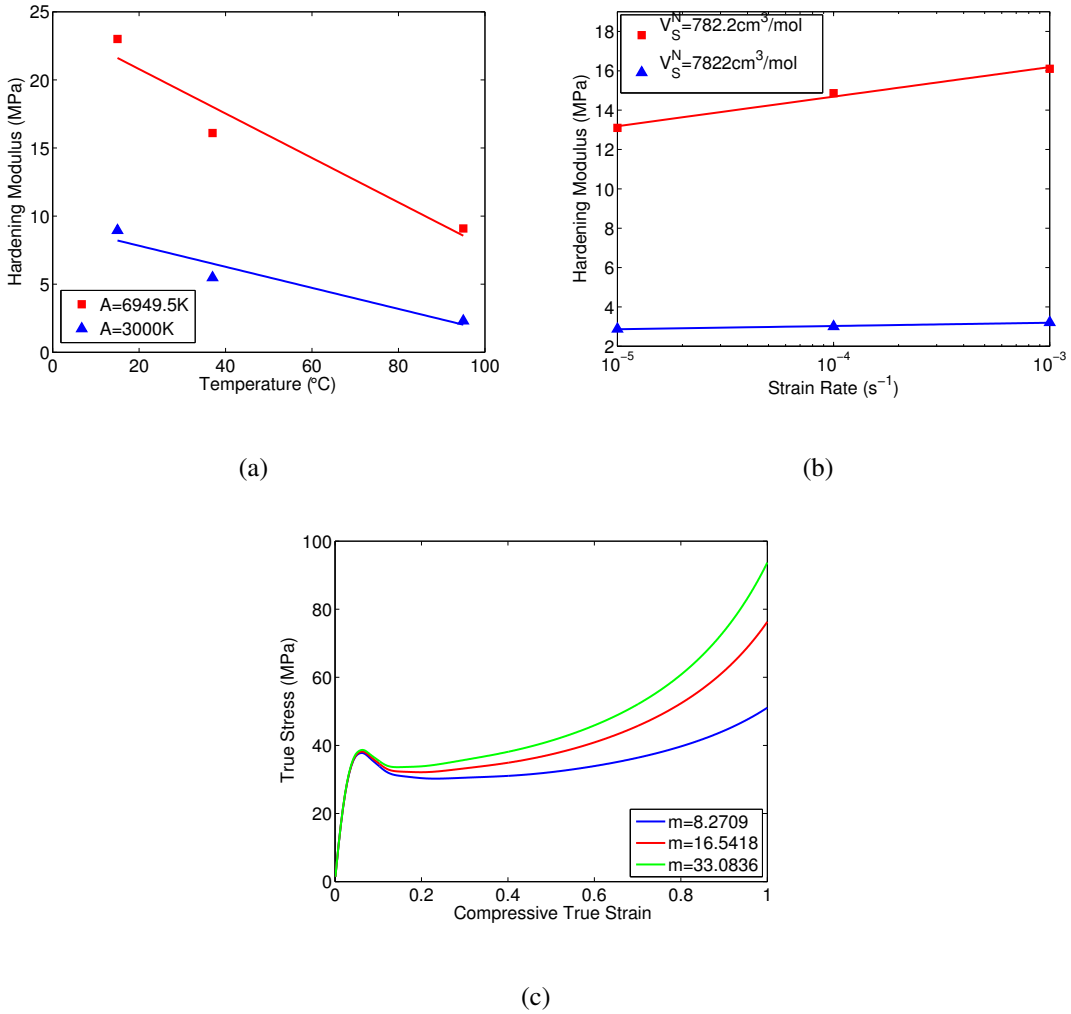


Figure 4.18: Effects of parameters that control the temperature, strain rate and strain state dependence of hardening (a) Hardening modulus at 15°C, 37°C and 95°C with strain rate of 10^{-3}/s, for $A = 6949.5K$ and $A = 3000K$ respectively. (b) Hardening modulus at 10^{-3}/s, 10^{-4}/s and 10^{-5}/s with temperature of 37°C, for $V_S^N = 782.2\text{cm}^3/\text{mol}$ and $V_S^N = 7822\text{cm}^3/\text{mol}$ respectively. (c) Stress response at 95°C and 10^{-3}/s, for $m = 8.2709$, $m = 16.5418$ and $m = 33.0836$ respectively.

4.6 Discussion

Comparing eq. (4.42) and eq. (4.43), for the viscous resistance to local segmental rearrangements we applied the Adam-Gibbs model in which the viscous resistance to local segmental rearrangements depends on both of the temperature and configurational entropy (or effective temperatures), while for the viscous resistance to network deformation we assumed that it only depends on the temperature. In fact, we explored the possibility of incorporating the configurational entropy in the expression of viscous resistance to network deformation,

$$\nu_j^N = \nu_j^{N,ref} \exp\left(\frac{B^N}{T\sum_i^Q \eta_i^c}\right)\left(\frac{\alpha_0-\alpha_c}{\alpha-\alpha_c}\right)\alpha^{-m}\frac{V_S^N S_{back}}{RT}\left[\sinh\left(\frac{V_S^N S_{back}}{RT}\right)\right]^{-1}, \tag{4.64}$$

where B^N is a thermal activation energy in analog to B in eq. (4.42). We noticed that this expression gives opposite rate dependence of strain hardening, which means the hardening modulus decreases with strain rate. This is inconsistent with experiment observations and thus unphysical. The reason for this opposite rate dependence is that a higher strain rate drives the system to a more disordered state with higher configurational entropy, as shown in Fig. 4.19(a), and from eq. (4.64), this leads to a lower viscous resistance to network deformation, as shown in Fig. 4.19(b). As a result, the model exhibits smaller hardening modulus at higher strain rate, as shown in Fig. 4.19(c). For this reason, we excluded the entropy term in the expression of viscous resistance to network deformation. This makes the network relaxation mechanism and the backstress independent from the effective tem-

peratures. Therefore, configurational contributions of the network resistance can affect the evolution of effective temperatures, but have little effects on the hardening stress response. This is demonstrated in Fig. 4.20, where we simulated the uniaxial compression response at 95°C and engineering strain rate of 3×10^{-4}/s. The contributions of network resistance were either all assigned in the kinematic subsystem or all assigned in the configurational subsystems. The stress response exhibited little difference. In Sec. 4.2.2.3, we assumed pure kinematic contributions. One can also assume pure configurational contributions, or in general, split the contributions into kinematic and configurational subsystems with a fraction. The stress response is insensitive to these assumptions.

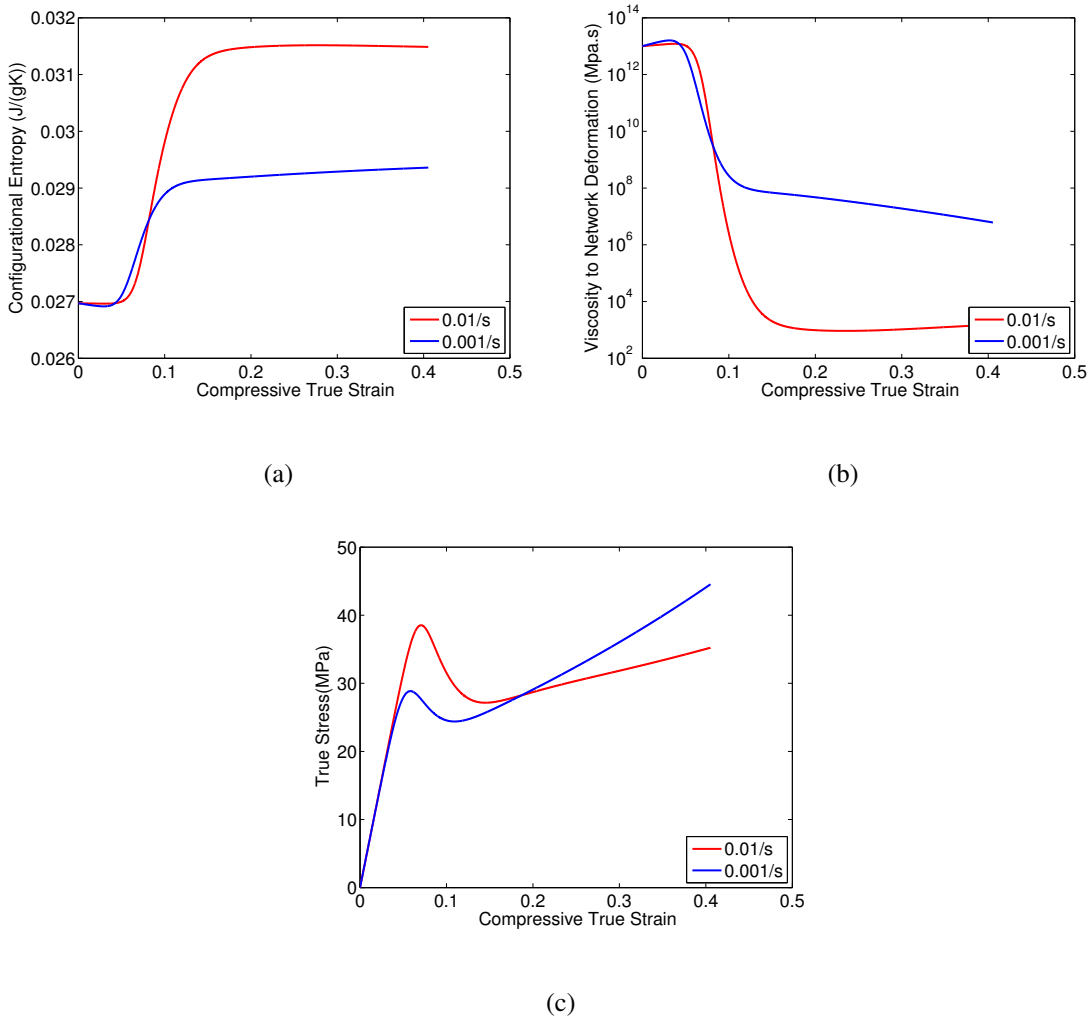


Figure 4.19: Incorporating configurational entropy in the viscous resistance to network deformation leads to opposite rate dependence of strain hardening, comparing the (a) evolution of configurational entropy, (b) evolution of viscous resistance to network deformation and (c) stress vs. strain for two strain rates, 10^{-2}/s and 10^{-3}/s.

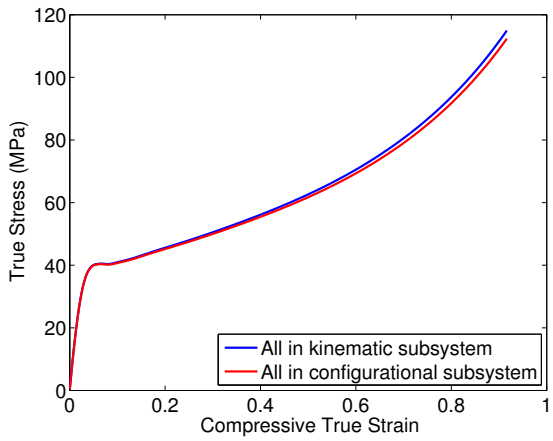


Figure 4.20: Comparison of the stress response between pure kinematic and pure configurational contributions of network resistance.

4.7 Conclusions

In conclusion, we extended the effective temperature theory in [82] to large strain and developed a constitutive model in a thermodynamically consistent manner. The model incorporates two mechanisms: the stretching and orientation of the polymer network, which leads to the development of a backstress, and network relaxation, which accounts for the temperature and rate dependence of strain hardening. The model was applied to simulate the thermomechanical behavior of polycarbonate. All of the model parameters were determined through standard thermomechanical tests. The simulation results showed good agreements with experiments and the temperature dependence, strain rate dependence and strain-dependence of hardening were quantitatively captured. In both experiments and simulation, the hardening modulus decreases with temperature and increases with logarithm

strain rate in an approximately linear manner. The model was also able to describe phenomena related to unloading and reloading, including the Bauschinger effect and the deformation induced anisotropy. In the constitutive model, the temperature parameter A, the activation volume for molecular relaxation V_S^N and the power of the orientation parameter m control the temperature dependence, the strain rate dependence and the strain dependence of hardening respectively.

Chapter 5

Conclusions and Future Work

This work has covered a wide range of active soft materials, from elastomers, hydrogels to glassy polymers. These materials are inherently different, in terms of both the mechanical properties and the stimuli they are responsive to. The Young's modulus varies from tens of kilopascals for hydrogels to 1-2 gigapascals for glassy polymers and the stimuli include temperature, solution and electric field etc. Despite these significant differences, however, this work has demonstrated that various kinds of soft materials can be studied with a similar set of methods. The developed methodology covers different stages of the study, from material characterization to constitutive modeling, from simulation to experiments.

Techniques including dynamic mechanical analysis (DMA) and differential scanning calorimetry (DSC) were applied in the material characterization. In Chapter 2 and Chapter 4, the viscoelasticity of dielectric elastomers and glassy polymers were characterized in a similar way. The stress relaxation spectrum was determined from either the master curve

of relaxation modulus versus time or the master curve of storage modulus versus frequency, both describing the viscoelasticity but from different aspects. The Time-Temperature Superposition (TTS) and the approximation to the continuous relaxation spectrum were utilized for both studies.

In terms of constitutive modeling, the state variable approach was applied for all studies in this work. The total free energy density was additively decomposed to a mechanical part and a part representing the contributions from physical stimuli. For dielectric elastomers, this part is the electrical energy; for hydrogels it's the mixing energy between polymer and solvent; for glassy polymers it's the thermal contributions in response to temperature change. Accordingly, state variables such as the electric field E, the solid fraction ϕ and the effective temperatures T_{e_i} were introduced to characterize the thermodynamic state of the materials. The first two are equilibrium state variables while the effective temperature is a nonequilibrium state variable and its evolution to equilibrium is affected by multiple mechanisms as discussed in Chapter 4. The coupling between these state variables and the mechanical deformation allows the developed models to capture various stimuli responsive behavior. In addition, the mechanical contributions can be elastic, as in the case of hydrogels, or viscoelastic as in the cases of dielectric elastomers and glassy polymers. For viscoelastic modeling, the inelastic stretches (or the inelastic deformation gradients in general three-dimensional case) were also introduced as state variables and evolve following certain flow rule.

Finite element analysis was applied in Chapter 3 and Chapter 4. The simulation in-

volves 3D or 2D axisymmetric geometries and boundary conditions in displacements or heat convection. Post-simulation analysis were addressed for both chapters. In Chapter 3, the strain energy was integrated over the body and the local Gaussian curvatures of deformed configurations were computed. In Chapter 4, the macroscopic heat flux was integrated and compared to experiment measurements.

Experiments were performed in all three chapters above. These include pure mechanical experiments, for example creep, stress relaxation and uniaxial compression, and also stimuli actuation experiments, as illustrated in Chapter 2 and Chapter 3. The experiment results were applied to calibrate the model as well as verify the model predictions.

5.1 Summary of this Work

Chapter 2 investigated the temperature dependent viscoelastic behavior of dielectric elastomers and the effects of viscoelasticity on the electro-actuation behavior. We measured the viscoelastic relaxation spectrum of VHB 4905 and the temperature dependence of the relaxation times and applied the results to a discrete multi-process viscoelastic model. We performed uniaxial creep and stress relaxation experiments and electrical actuation experiments under different prestretch conditions. The measured spectrum was applied to predict the experimental results. Generally, the model produced good quantitative agreement with both the viscoelastic and electro-actuation experiments. We showed that the viscoelastic spectrum can be truncated systematically to describe the time-dependent behavior in a more

narrow time range, though the truncation was shown to require multiple, and not a single nonequilibrium process in order to accurately capture the viscoelastic response. Moreover, the spectrum can be shifted using the temperature-dependent shift factor to describe the time-dependent behavior at higher temperatures. We also investigated the failure of VHB membrane caused by viscoelastic creep when prestretched and subjected to constant voltage loading. The experimental time to failure for the specimens decreased exponentially with voltage, which agreed well with the predictions of the model.

In Chapter 3, we designed a thermoresponsive self-folding system composed of soft swelling gels and stiff non-swelling segments arranged in a regular grid pattern. The self-folding behavior of the composite plate was investigated using experiments and finite element modeling. In both experiments and simulation, the system exhibited bidirectional and biaxial bending. Cooling the patterned bilayer structure leads to bending upwards about one axis, while heating leads to bending downwards about a different axis 90^o from the first. The direction of bending was determined by the distribution of the swelling ratio through the thickness of the composite plate and can be altered by imparting a swelling gradient through the thickness. The choice of bending axis was explained through comparing the energy of the different equilibrium folded configurations. Biaxial bending occurred when the segments were arranged such that the long axis of the patch was perpendicular to the long axis of the plate. The curvature of the bent plate was controlled by the segment spacing, and a large segment spacing led to the development of wrinkles during bending.

In Chapter 4, we developed a constitutive model for glassy polymers based on the ef-

fective temperature theory that is capable of describing the strain hardening behavior at large deformation. The model incorporates two mechanisms: one represents the stretching and orientation of the polymer network, which leads to the development of a backstress; the other one represents the network relaxation, which accounts for the temperature and rate dependence of strain hardening. The model was implemented into finite element programs and was applied to simulate the thermomechanical behavior of polycarbonate. All of the model parameters were determined through standard thermomechanical tests. The simulation results showed good agreements with experiments and the temperature dependence, strain rate dependence and strain state dependence of hardening were quantitatively captured.

5.2 Limitations of this Work

Though the developed models have achieved considerable success in describing the coupled physical-mechanical behavior of active soft materials, some limitations of this work should be acknowledged. In Chapter. 2, the developed constitutive model assumed that the dielectric elastomer is ideal and its permittivity is constant. The performed simulations are material point simulations, in which the complex boundary conditions in actuation experiments were neglected. This led to the discrepancy in actuation stretch between experiments and simulation, as shown in Fig. 2.9. In Chapter. 3, the applied constitutive model for hydrogels is an equilibrium model, which is not capable of simulating the diffu-

sion process. The finite element simulations assumed a uniformly distributed temperature field, which neglected the heat conduction in experiments. The simulations also assumed an idealized, flat, and stress-free initial state, while the plates tended to exhibit a saddle shape near the LCST in the experiments. In Chapter 4, the developed constitutive model also made a number of assumptions. The theory neglects thermal interactions between the configurational subsystems and assumed that the relaxation time and the viscosity of each structural and stress relaxation process shared same dependence on the temperature and nonequilibrium structure. The model also assumed that the configurational properties are identically distributed over the configurational subsystems, thus can be described by the same structural relaxation spectrum. Moreover, we assumed that the network relaxation spectrum can be obtained through shifting of the stress relaxation spectrum. In addition, we neglected the effects of temperature changes $\dot{T}$ in the simulations. This assumption may be valid for the low strain rates discussed in Chapter 4, but should be included for general cases.

5.3 Future Directions

Future directions include further improvements on current models and the applications of developed models in simulating new experimental observations.

The constitutive model developed for dielectric elastomers needs to be implemented into finite element programs. This will allow the model to simulate various kinds soft

robotic actuators made from dielectric elastomers. Also, novel actuator designs with backgrounds in industrial applications need be proposed and the corresponding actuation experiments need to be performed to further explore the design space of DE.

In Chapter 3, the applied equilibrium model for hydrogels can be improved by incorporating the diffusion mechanism. This will allow the model to predict the nonequilibrium configurations that evolve with time. The design space of the bilayer plates with soft and stiff segments can be further explored. For example, the shape of the plates and segments can be triangular or circular instead of being rectangular as discussed in Chapter 3. Also, the segments can be noneunionormly distributed over the plate and the pattern of distribution will have effects on deformation.

The polycarbonate filaments used in Chapter 4 are widely used as raw material for fused filament fabrication (FFF), which is a kind of rapidly growing additive manufacturing (AM) process. Therefore, the developed constitutive model and the measured material parameters can be applied to simulate the thermomechanical behavior of printed welds and other structures. The simulation results will play an important role in guiding the design and manufacturing process. The constitutive model based on effective temperature theory also needs to be further investigated, for example in terms of the ability to predict the heat generation at different strain rates. In addition, the model can be improved by incorporating the diffusion of effective temperatures to simulate the development of heterogeneous strain fields and strain localization.

Appendix A

Incorporating Thermal Deformation in the Effective Temperature Theory

In this appendix, we incorporate the thermal deformation into the constitutive model developed in Chapter 4. The method for determining additional parameters is also included.

A.1 Kinematics

The total deformation gradient $\mathbf{F}$ is split multiplicatively into a thermal component $\mathbf{F}_T = J_T^{1/3}\mathbf{I}$ and a mechanical component $\mathbf{F}_M$ [239, 240],

$$\mathbf{F} = J_T^{1/3}\mathbf{F}_M. \tag{A.1}$$

APPENDIX A. INCORPORATING THERMAL DEFORMATION IN THE EFFECTIVE TEMPERATURE THEORY

To model the structural relaxation process, we decompose the thermal deformation into an instantaneous part given by $(T - T_0)$ and a departure from the instantaneous response given by $(\sum_i^Q \phi_i T_{e_i} - T_0)$,

$$J_T = \exp[\alpha_g (T - T_0)] \exp[\Delta\alpha (\sum_i^Q \phi_i T_{e_i} - T_0)], \tag{A.2}$$

where α_g is the glassy volumetric coefficient of thermal expansion (CTE), α_r is the rubbery volumetric CTE, and $\Delta\alpha = \alpha_r - \alpha_g$. The mechanical component $\mathbf{F}_M$ is split multiplicatively into P parallel elastic and inelastic components, as expressed in eq. (4.10),

$$\mathbf{F}_M = \mathbf{F}_k^e \mathbf{F}_k^v, \text{for} k = 1...P, \tag{A.3}$$

and the inelastic deformation gradient $\mathbf{F}_k^v$ is further decomposed into two parts representing the network deformation and relaxation components, same as eq. (4.12),

$$\mathbf{F}_k^v = \mathbf{F}_k^{N,e} \mathbf{F}_k^{N,v}, \text{for} k = 1...P. \tag{A.4}$$

A.2 Constitutive model

The constitutive model can be developed through the same thermodynamics derivation in Sec. 4.2.2.3. We apply the same form of Helmholtz free energy density as eq. (4.38),

$$\begin{aligned}\Psi^k &= \sum_j^P \frac{(1-a)\mu_j^{neq}}{2}(\mathrm{tr}(\bar{\mathbf{C}}_j^e)-3) + \frac{\kappa}{4}({J_M}^2 - 2\log J_M - 1) \\ &\quad + c_{g0}(T-T_0) - c_{g0}T\ln\frac{T}{T_0} - \frac{c_{g1}}{2}(T-T_0)^2 + \sum_j^P \xi(\overline{\mathbf{b}_j^{N,e}}), \\ \Psi_i^c &= \frac{T_{e_i}}{T_0}\frac{\mu_i^{eq}}{2}(\mathrm{tr}(\bar{\mathbf{C}}_M)-3) + \sum_j^P \frac{a_i\mu_j^{neq}}{2}(\mathrm{tr}(\bar{\mathbf{C}}_j^e)-3) + \Delta c_{0_i}(T_{e_i}-T_0) - \Delta c_{0_i}T_{e_i}\ln\frac{T_{e_i}}{T_2} - \frac{\Delta c_{1_i}}{2}(T_{e_i}-T_2)^2,\end{aligned} \tag{A.5}$$

where $J_M = \det\mathbf{F}_M$ is the mechanical part of volumetric deformation and $\overline{\mathbf{C}_M} = J_M^{-2/3}\mathbf{C}_M = J_M^{-2/3}(\mathbf{F}_M^T\mathbf{F}_M)$ is the isochoric part of the mechanical deformation tensor. The second Piola-Kirchhoff stress can be evaluated as $\mathbf{S} = 2\frac{\partial\psi}{\partial\mathbf{C}}$ and the Cauchy stress, $\sigma = \frac{1}{J}\mathbf{F}\mathbf{S}\mathbf{F}^T$ is the push-forward of $\mathbf{S}$,

$$\sigma = \underbrace{\frac{1}{J}(\sum_i^Q \frac{T_{e_i}\phi_i}{T_0})\mu^{eq}(\overline{\mathbf{b}_M} - \frac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_M})\mathbf{I})}_{\sigma^{eq}} + \sum_j^P \underbrace{(\frac{1}{J}\mu_j^{neq}(\overline{\mathbf{b}_j^e} - \frac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_j^e})\mathbf{I})}_{\sigma_j^{neq}} + \underbrace{\frac{1}{2J}\kappa({J_M}^2-1)}_{p}\mathbf{I}. \tag{A.6}$$

The expressions for stress tensors $\mathbf{M}_j$, $\mathbf{N}_j{}^{back}$ and ζ_j and the flow rules remain unchanged,

$$\begin{aligned}\mathbf{M}_j &= \sum_i^Q \phi_i(1+a(\tfrac{T}{T_{e_i}}-1))\mu_j^{neq}(\overline{\mathbf{C}}_j^e - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{C}}_j^e)\mathbf{I}), \\ \mathbf{N}_j^{back} &= \mu_j^{back}\frac{\lambda_L}{\bar{\lambda}_{eff,j}}\mathscr{L}^{-1}(\tfrac{\bar{\lambda}_{eff,j}}{\lambda_L})(\overline{\mathbf{b}_j^{N,e}} - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{b}_j^{N,e}})\mathbf{I}), \\ \zeta_j &= \mathbf{M}_j - \mathbf{N}_j^{back}.\end{aligned} \tag{A.7}$$

$$\mathbf{D}_j^{\nu} = \frac{1}{2\nu_j^I}\zeta_j, \nu_j^I = \nu_j^{I,ref}\exp(\frac{B}{T\sum\limits_i^Q \eta_i^c})\frac{V_S^I S}{RT}[\sinh(\frac{V_S^I S}{RT})]^{-1}, \tag{A.8}$$

$$\mathbf{D}_j^{N,\nu} = \frac{1}{2\nu_j^N}\mathbf{N}_j{}^{back}, \nu_j^N = \nu_j^{N,ref}\exp(\frac{A}{T})(\frac{\alpha_0-\alpha_c}{\alpha-\alpha_c})\alpha^{-m}\frac{V_S^N S^{back}}{RT}[\sinh(\frac{V_S^N S^{back}}{RT})]^{-1}. \tag{A.9}$$

The configurational entropy can be evaluated from eq. (4.24),

$$\begin{aligned}\eta_i^c &= \Delta c_0\phi_i\log\tfrac{T_{e_i}}{T_2} + \Delta c_1\phi_i(T_{e_i}-T_2)\\ &-\tfrac{\mu^{eq}\phi_i}{2T_0}(\mathrm{tr}(\overline{\mathbf{C}_M})-3) + \tfrac{\kappa}{2}(J^2{}_M-1)\Delta\alpha\phi_i.\end{aligned} \tag{A.10}$$

Substituting eq. (A.10) into eq.(4.37) and neglecting the diffusion term gives the evolution equation for effective temperatures,

$$\begin{aligned}\dot{T}_{e_i} &= \tfrac{T-T_{e_i}}{\tau_{R_i}} + \tfrac{a}{\Delta c_0+\Delta c_1 T_{e_i}}\sum_j^P \mu_j^{neq}(\overline{\mathbf{C}}_j^e - \tfrac{1}{3}\mathrm{tr}(\overline{\mathbf{C}}_j^e)\mathbf{I}) : \mathbf{D}_j^{\nu}\\ &+\tfrac{J^{-2/3}T_{e_i}\mu^{eq}}{2(\Delta c_0+\Delta c_1 T_{e_i})T_0}(\mathbf{I}-\tfrac{1}{3}\mathrm{tr}(\mathbf{C}_M)\mathbf{C}_M^{-1}) : \dot{\mathbf{C}} - \tfrac{T_{e_i}\kappa\Delta\alpha J_M^2}{\Delta c_0+\Delta c_1 T_{e_i}}\tfrac{\dot{J}_M}{J_M}.\end{aligned} \tag{A.11}$$

The last terms in eq. (A.10) and eq. (A.11) are additional terms that come from contributions of the thermal deformation.

A.3 Measurement of the CTEs for polycarbonate

The constitutive model in Sec. A.2 has two additional parameters compared to the original model in Chapter 4, which are the glassy volumetric CTE α_g and the rubbery volumetric CTE α_r. To measure these two parameters for the polycarbonate, constant cooling rate tests were performed on the film specimens under uniaxial tension using a TA Q800 Dynamic Mechanical Analyzer (DMA) set to zero force mode. The specimens were equilibrated at 140°C and cooled to 60°C at 1°C/min. The experiments measured the thermal contraction in length, defined as the change in length over the original length, as a function of temperature. The rubbery volumetric CTE α_r was determined as three times the slope of the curve between 106°C and 139°C, while the glassy volumetric CTE α_g was determined as three times the slope of the curve between 67°C and 75°C, as shown in Fig. A.1 [229, 230]. The measured values are $\alpha_g = 1.7731 \times 10^{-4}$/K and $\alpha_r = 5.6214 \times 10^{-4}$/K.

APPENDIX A. INCORPORATING THERMAL DEFORMATION IN THE EFFECTIVE TEMPERATURE THEORY

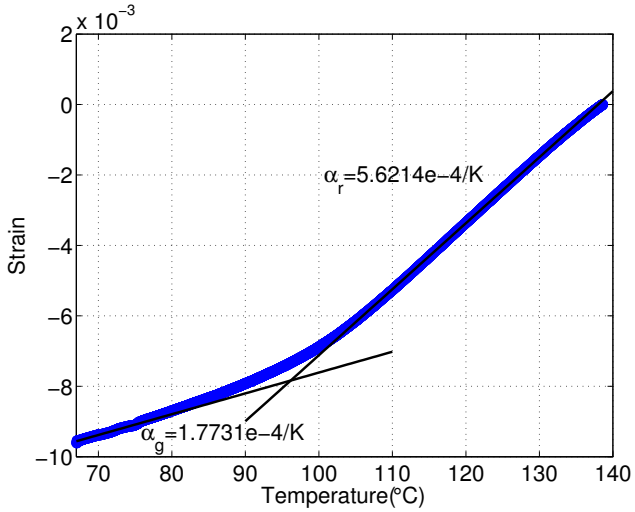


Figure A.1: Experimental data of the thermal strain for cooling rate of 1°C/min. Solid black lines indicate linear fits used to determine the CTEs.

Appendix B

Finite Element Implementation of the Constitutive Model Based on Effective Temperature Theory

In this appendix, the constitutive model introduced in Chapter 4 is implemented as a user-defined material subroutine into finite element code. The algorithm is summarized first and the C++ program is attached.

B.1 Algorithm

B.1.1 Solving the Internal Variables

B.1.1.1 Evolution of the Effective Temperatures

From eq. (4.37), the evolution equation for effective temperatures can be written as

$$\dot{T}_{e_i} = -\frac{1}{\tau_{R_i}}(T_{e_i} - T) + \frac{1}{\Delta c_i}\sum_{j}^{N} 2\mathbf{C}_j^e \frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^v - \frac{T_{e_i}}{\Delta c_i} 2\frac{\partial \eta_i^c}{\partial \mathbf{C}} : \frac{1}{2}\dot{\mathbf{C}}, \tag{B.1}$$

We define the following coefficients,

$$\begin{aligned} \theta_i^1 &= \tfrac{1}{\Delta c_i}, \\ \theta_i^2 &= \tfrac{a}{\Delta c_i}. \end{aligned} \tag{B.2}$$

The plastic work and latent heat are defined as,

$$W^{plastic} = \frac{\Delta t}{a_i}\sum_{j}^{P} 2\mathbf{C}_j^e \frac{\partial \Psi_i^c}{\partial \mathbf{C}_j^e} : \mathbf{D}_j^v = \sum_{j}^{P} \mathbf{C}_j^e \mathbf{S}_j^{neq} : (\mathbf{D}_j^v \Delta t), \tag{B.3}$$

$$\begin{aligned} W_i^{latent} &= -\tfrac{T_{e_i}\Delta t}{\phi_i} 2\frac{\partial \eta_i^c}{\partial \mathbf{C}} : \tfrac{1}{2}\dot{\mathbf{C}} = \tfrac{T_{e_i}\Delta t}{\phi_i} 2\frac{\partial^2 \Psi}{\partial T_{e_i}\partial \mathbf{C}} : \tfrac{1}{2}\dot{\mathbf{C}} = \tfrac{T_{e_i}\Delta t}{\phi_i} 2\frac{\partial \mathbf{S}}{\partial T_{e_i}} : \tfrac{1}{2}\dot{\mathbf{C}} \\ &= \tfrac{T_{e_i}\Delta t}{\phi_i} 2\frac{\partial \mathbf{S}^{\mathrm{eq}}}{\partial T_{e_i}} : \tfrac{1}{2}\dot{\mathbf{C}} = \frac{T_{e_i}}{\sum_i^Q T_{e_i}\phi_i} J\sigma^{eq} : \mathbf{D}\Delta t. \end{aligned} \tag{B.4}$$

With eqs. (B.2), (B.3) and (B.4), the evolution equation (B.1) can be discretized as,

$$\frac{T_{e_i}^n - T_{e_i}^p}{\Delta t} = \frac{T - T_{e_i}^n}{\tau_{R_i}} + \theta_i^2 \frac{W^{plastic}}{\Delta t} + \theta_i^1 \frac{W_i^{latent}}{\Delta t}, \tag{B.5}$$

where $T_{e_i}^n$ and $T_{e_i}^p$ are the effective temperatures at current time step and previous time step respectively. Define the residual as,

$$r_{T_{e_i}} = T_{e_i}^n + \frac{\Delta t}{\tau_{R_i}}(T_{e_i}^n - T) - \theta_i^2 W^{plastic} - \theta_i^1 W_i^{latent} - T_{e_i}^p. \tag{B.6}$$

We can organize the residuals of Q processes into a vector, $\left(\mathbf{r}_{\mathbf{T_e}}\right)_i = r_{T_{e_i}}$, for $i = 1...Q$.

B.1.1.2 Evolution of the Network Elastic Strain

From Chapter 4, the flow rule for network relaxation can be written as,

$$\mathbf{D}_j^{N,\nu} = \frac{1}{2\nu_j^N} \mathbf{N}_j{}^{back}. \tag{B.7}$$

Following [120], express the above equation in principle coordinates,

$$\varepsilon_{i,j}^{N,e} - \varepsilon_{i,j(tr)}^{N,e} = -\frac{\Delta t}{2\nu_j^N} N_{i,j}^{back}, \quad \text{for} \quad i = 1,2,3 \quad \text{and} \quad j = 1...P, \tag{B.8}$$

where $\varepsilon_{i,j}^{N,e}$ and $\varepsilon_{i,j(tr)}^{N,e}$ are the principle network elastic strains in current state and trial state, and $N_{i,j}^{back}$ are the principle back stresses. Define the residuals as

$$r_{\varepsilon_{i,j}}^{N} = \varepsilon_{i,j}^{N,e} + \frac{\Delta t}{2\nu_j^N} N_{i,j}^{back} - \varepsilon_{i,j(tr)}^{N,e}. \tag{B.9}$$

Organize the residuals into a vector with $3P$ elements,

$$\mathbf{r}_{\varepsilon}^{\mathbf{N}} = \begin{pmatrix} r_{\varepsilon_{1,1}}^{N} \\ r_{\varepsilon_{2,1}}^{N} \\ r_{\varepsilon_{3,1}}^{N} \\ \vdots \\ r_{\varepsilon_{1,P}}^{N} \\ r_{\varepsilon_{2,P}}^{N} \\ r_{\varepsilon_{3,P}}^{N} \end{pmatrix}. \tag{B.10}$$

B.1.1.3 Evolution of the Elastic Strain

From Chapter 4, the flow rule for inelastic deformation can be written as,

$$\mathbf{D}_j^v = \frac{1}{2\nu_j^I}(\mathbf{M}_j - \mathbf{N}_j^{back}). \tag{B.11}$$

In order to express the above equation in principle coordinates, we need the Mandel stress $\mathbf{M}_j$ and the back stress $\mathbf{N}_j^{back}$ to be coaxial. We prove here $\mathbf{M}_j$ and $\mathbf{N}_j^{back}$ are coaxial if all

the rotation effects are lumped into the inelastic and network inelastic deformation.

From polar decomposition, the kinematics law (4.10) can be further decomposed into

$$\mathbf{F} = \mathbf{F}^e\mathbf{F}^v = \mathbf{V}^e\mathbf{R}^e\mathbf{R}^v\mathbf{U}^v = \mathbf{V}^e\mathbf{R}\mathbf{U}^v, \tag{B.12}$$

where we neglected the subscripts notating different processes for simplicity. As shown in eq. (B.12), the total rotation tensor $\mathbf{R}$ can be decomposed into the product of elastic and plastic rotation tensors $\mathbf{R} = \mathbf{R}^e\mathbf{R}^v$. However, this decomposition is indeterminate and different choices can be made without affecting the material response, as discussed by Boyce et al. [241] Therefore, we can assign all the rotation effects in the plastic deformation, i.e. $\mathbf{R}^v = \mathbf{R}$ and $\mathbf{R}^e = \mathbf{I}$. This results in a symmetric and therefore, unique elastic deformation gradient, $\mathbf{F}^{e^T} = \mathbf{F}^e$. The same idea can be applied to the decomposition of the inelastic deformation gradient,

$$\mathbf{F}^v = \mathbf{F}^{N,e}\mathbf{F}^{N,v} = \mathbf{V}^{N,e}\mathbf{R}^{N,e}\mathbf{R}^{N,v}\mathbf{U}^{N,v} = \mathbf{V}^{N,e}\mathbf{R}^v\mathbf{U}^{N,v}. \tag{B.13}$$

Again, we assign all the rotation effects in the network inelastic deformation, such that $\mathbf{R}^{N,v} = \mathbf{R}^v = \mathbf{R}$ and $\mathbf{R}^{N,e} = \mathbf{I}$. With this assignment, the network elastic deformation gradient is also symmetric and unique, $\mathbf{F}^{N,e^T} = \mathbf{F}^{N,e}$. The sequence of deformation maps produced by the successive decompositions of the deformation gradient and the assignment of the rotation tensor are illustrated in Fig. B.1.

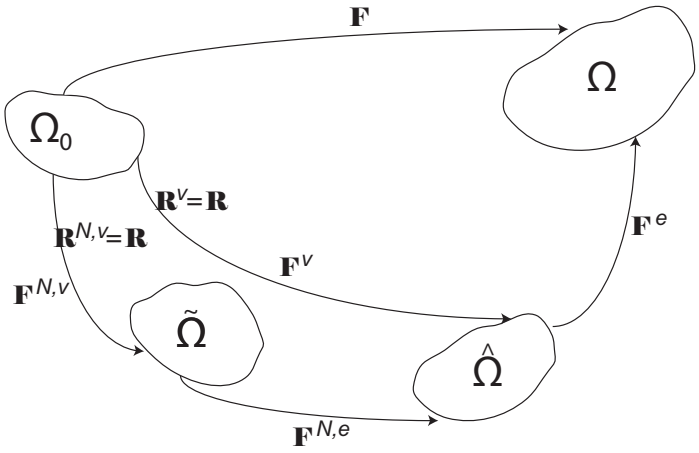


Figure B.1: Sequence of deformation maps produced by the successive decompositions of the deformation gradient and the assignment of the rotation tensor.

From Fig. B.1, it's straightforward to see that the product of $\mathbf{F}^e$ and $\mathbf{F}^{N,e}$ is also symmetric and unique, $(\mathbf{F}^{N,e}\mathbf{F}^e)^T = \mathbf{F}^{N,e}\mathbf{F}^e$, because all the rotation effects are assigned in the network inelastic deformation $\mathbf{R}^{N,v} = \mathbf{R}$. Therefore,

$$\begin{aligned} &\mathbf{F}^{N,e}\mathbf{F}^e = \mathbf{F}^{N,e^T}\mathbf{F}^{e^T} = (\mathbf{F}^e\mathbf{F}^{N,e})^T = (\mathbf{F}^{N,e}\mathbf{F}^e)^T \\ &\Rightarrow \mathbf{F}^{N,e}\mathbf{F}^e = \mathbf{F}^e\mathbf{F}^{N,e}, \end{aligned} \tag{B.14}$$

which is equivalent to the coaxiality between symmetric tensors $\mathbf{F}^e$ and $\mathbf{F}^{N,e}$. For isotropic material, this leads to the conclusion that $\mathbf{M}_j$ and $\mathbf{N}_j^{back}$ are coaxial. Given this, we can express eq. (B.11) in principle coordinates in a similar way with Sec. B.1.1.2,

$$\varepsilon^e_{i,j} - \varepsilon^e_{i,j(tr)} = -\frac{\Delta t}{2\nu^I_j}(M_{i,j} - N^{back}_{i,j}), \quad \text{for} \quad i = 1,2,3 \quad \text{and} \quad j = 1...P, \tag{B.15}$$

where $\varepsilon^e_{i,j}$ and $\varepsilon^e_{i,j(tr)}$ are the principle elastic strains in current state and trial state, and $M_{i,j}$

are the principle Mandel stresses. Define the residuals as

$$r^I_{\varepsilon_{i,j}} = \varepsilon^e_{i,j} + \frac{\Delta t}{2\nu^I_j}(M_{i,j} - N^{back}_{i,j}) - \varepsilon^e_{i,j(tr)}. \tag{B.16}$$

Organize the residuals into a vector with $3P$ elements,

$$\mathbf{r}^{\mathbf{I}}_{\boldsymbol{\varepsilon}} = \begin{pmatrix} r^I_{\varepsilon_{1,1}} \\ r^I_{\varepsilon_{2,1}} \\ r^I_{\varepsilon_{3,1}} \\ \vdots \\ r^I_{\varepsilon_{1,P}} \\ r^I_{\varepsilon_{2,P}} \\ r^I_{\varepsilon_{3,P}} \end{pmatrix}. \tag{B.17}$$

B.1.1.4 Numerical Solution Procedure

There are three groups of independent internal variables that need to be solved from eqs. (B.5), (B.8) and (B.15): the effective temperatures, the network elastic strains and the

elastic strains. The vector form of these variables are defined as,

$$\mathbf{T_e} = \begin{pmatrix} T_{e_1} \\ T_{e_2} \\ \vdots \\ T_{e_{Q-1}} \\ T_{e_Q} \end{pmatrix}, \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{N} = \begin{pmatrix} \varepsilon_{1,1}^{N,e} \\ \varepsilon_{2,1}^{N,e} \\ \varepsilon_{3,1}^{N,e} \\ \vdots \\ \varepsilon_{1,P}^{N,e} \\ \varepsilon_{2,P}^{N,e} \\ \varepsilon_{3,P}^{N,e} \end{pmatrix}, \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{I} = \begin{pmatrix} \varepsilon_{1,1}^{e} \\ \varepsilon_{2,1}^{e} \\ \varepsilon_{3,1}^{e} \\ \vdots \\ \varepsilon_{1,P}^{e} \\ \varepsilon_{2,P}^{e} \\ \varepsilon_{3,P}^{e} \end{pmatrix}. \tag{B.18}$$

Define the following tangent matrices,

$$\begin{aligned} &\mathbf{K}_{\mathbf{T_e}} = \frac{\partial \mathbf{r}_{\mathbf{T_e}}}{\partial \mathbf{T_e}}, \mathbf{K}_{\mathbf{T_e a}} = \frac{\partial \mathbf{r}_{\mathbf{T_e}}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{N}}, \mathbf{K}_{\mathbf{T_e r}} = \frac{\partial \mathbf{r}_{\mathbf{T_e}}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{I}}; \\ &\mathbf{K}_{\mathbf{a T_e}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{N}}{\partial \mathbf{T_e}}, \mathbf{K}_{\mathbf{a}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{N}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{N}}, \mathbf{K}_{\mathbf{ar}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{N}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{I}}; \\ &\mathbf{K}_{\mathbf{r T_e}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{I}}{\partial \mathbf{T_e}}, \mathbf{K}_{\mathbf{ra}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{I}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{N}}, \mathbf{K}_{\mathbf{r}} = \frac{\partial \mathbf{r}_{\boldsymbol{\varepsilon}}^\mathbf{I}}{\partial \boldsymbol{\varepsilon}_\mathbf{e}^\mathbf{I}}, \end{aligned} \tag{B.19}$$

and

$$\begin{aligned} \mathbf{K}_1 &= \mathbf{K}_\mathbf{a} - \mathbf{K}_{\mathbf{a T_e}} \mathbf{K}_{\mathbf{T_e}}^{-1} \mathbf{K}_{\mathbf{T_e a}}, \\ \mathbf{K}_2 &= \mathbf{K}_\mathbf{ar} - \mathbf{K}_{\mathbf{a T_e}} \mathbf{K}_{\mathbf{T_e}}^{-1} \mathbf{K}_{\mathbf{T_e r}}, \\ \mathbf{K}_3 &= \mathbf{K}_{\mathbf{T_e r}} - \mathbf{K}_{\mathbf{T_e a}} \mathbf{K}_\mathbf{a}^{-1} \mathbf{K}_\mathbf{ar}, \\ \mathbf{K}_4 &= \mathbf{K}_{\mathbf{T_e}} - \mathbf{K}_{\mathbf{T_e a}} \mathbf{K}_\mathbf{a}^{-1} \mathbf{K}_{\mathbf{a T_e}}. \end{aligned} \tag{B.20}$$

The internal variables were solved by the Newton-Raphson method with a staggered scheme.

The algorithm is summarized below.

APPENDIX B. FINITE ELEMENT IMPLEMENTATION OF THE CONSTITUTIVE MODEL BASED ON EFFECTIVE TEMPERATURE THEORY

1. Iterate the effective temperatures $\mathbf{T_e}$ with

$$\begin{aligned} \Delta\mathbf{T_e}|_i &= -\mathbf{K}_{\mathbf{T_e}}^{-1}|_i \, \mathbf{r}_{\mathbf{T_e}}|_i \,, \\ \mathbf{T_e}|_{i+1} &= \mathbf{T_e}|_i + \Delta\mathbf{T_e}|_i \,, \end{aligned} \tag{B.21}$$

   until $\|\mathbf{r}_{\mathbf{T_e}}\| < tol$.

2. Update the network elastic strain $\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}$ with

$$\begin{aligned} \Delta\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}|_i &= -\mathbf{K}_1^{-1}|_i \, \mathbf{r}_{\varepsilon}^{\mathbf{N}}|_i \,, \\ \boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}|_{i+1} &= \boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}|_i + \Delta\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}|_i \,. \end{aligned} \tag{B.22}$$

3. Repeat step 1 with updated $\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{N}}$.

4. Repeat step 2 and step 3 until both $\|\mathbf{r}_{\mathbf{T_e}}\| < tol$ and $\left\|\mathbf{r}_{\varepsilon}^{\mathbf{N}}\right\| < tol$.

5. Update the elastic strain $\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}$ with

$$\begin{aligned} \Delta\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}|_i &= -(\mathbf{K_r} - \mathbf{K_{ra}}\mathbf{K}_1^{-1}\mathbf{K}_2 - \mathbf{K}_{\mathbf{rT_e}}\mathbf{K}_4^{-1}\mathbf{K}_3)^{-1}|_i \, \mathbf{r}_{\varepsilon}^{\mathbf{I}}|_i \,, \\ \boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}|_{i+1} &= \boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}|_i + \Delta\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}|_i \,. \end{aligned} \tag{B.23}$$

6. Repeat step 1-4 with updated $\boldsymbol{\varepsilon}_{\mathbf{e}}^{\mathbf{I}}$.

7. Repeat step 5 and step 6 until $\|\mathbf{r}_{\mathbf{T_e}}\| < tol$, $\left\|\mathbf{r}_{\varepsilon}^{\mathbf{N}}\right\| < tol$ and $\left\|\mathbf{r}_{\varepsilon}^{\mathbf{I}}\right\| < tol$.

In the program presented in Sec. B.2, step 1 was performed by the function *Compute_Tei*, step 2-4 were performed by the function *Compute_a* and step 5-7 were performed by the

function *Compute_le*.

B.1.2 Finite Element Implementation

The strong form of the balance of momentum can be written as,

$$\begin{aligned} &\mathrm{Div}(\mathbf{FS}) + \mathbf{B} = \mathbf{0}, \quad \text{in} \quad \Omega, \\ &\mathbf{U} = \widehat{\mathbf{U}}, \quad \text{on} \quad \Omega_u, \\ &\mathbf{T} = \mathbf{FS}.\mathbf{N} = \widehat{\mathbf{T}}, \quad \text{on} \quad \Omega_t. \end{aligned} \tag{B.24}$$

With the weighting function defined as $\mathbf{W}$, the corresponding weak form of the above equation is

$$\int \nabla \mathbf{W} : (\mathbf{FS}) dV = \int \mathbf{W}.\mathbf{B} dV + \int \mathbf{W}.\mathbf{T}_n dS. \tag{B.25}$$

The following shape functions and trial functions are defined,

$$\mathbf{W} = \sum N_A \mathbf{W}_A, \mathbf{U} = \sum N_A \mathbf{U}_A \tag{B.26}$$

Substituting eq. (B.26) into the weak form yields,

$$\int (\mathbf{FS}) \frac{\partial N_A}{\partial \mathbf{X}} dV = \int N_A \mathbf{B} dV + \int N_A \mathbf{T}_n dS. \tag{B.27}$$

Define the residual as,

$$\mathbf{R}_A^{\mathbf{U}} = \int (\mathbf{FS}) \frac{\partial N_A}{\partial \mathbf{X}} dV - \int N_A \mathbf{B} dV - \int N_A \mathbf{T}_n dS. \tag{B.28}$$

The tangent can be obtained through

$$\mathbf{K}_{AB}^{\mathbf{UU}} = \frac{\partial \mathbf{R}_A^{\mathbf{U}}}{\partial \mathbf{U}_B}. \tag{B.29}$$

Note that the residual in eq. (B.28) can be written in the index form,

$$R_A^{U_i} = \int F_{ij} S_{jk} \frac{\partial N_A}{\partial X_k} dV - \int N_A B_i dV - \int N_A T_{ni} dS. \tag{B.30}$$

Therefore the elements of $\mathbf{K}_{AB}^{\mathbf{UU}}$ can be evaluated as,

$$K_{AB}^{U_i U_n} = \int \frac{\partial N_A}{\partial X_k} \frac{\partial N_B}{\partial X_j} S_{jk} \delta_{in} dV + \int \frac{\partial N_A}{\partial X_k} \frac{\partial N_B}{\partial X_m} F_{ij} F_{nl} \mathbb{C}_{jkml} dV, \tag{B.31}$$

where $\mathbb{C}_{jkml} = \frac{\partial \mathbf{S}_{jk}}{\partial \mathbf{C}_{ml}}$. Several numerical methods have been developed to calculate the tangent moduli $\mathbb{C}_{jkml}$. In this implementation, we adopted the method developed by Reese and Govindjee (RG split method) [120]. The $\mathbb{C}_{jkml}$ is calculated in the function c_ijkl in the program in Sec. B.2.

B.2 Programs in C++

The model was implemented in an open-source finite element program Tahoe©(Sandia National Laboratories) [2]. The complete codes in C++ are attached below.

Head file: ET-multi.h

```
#ifndef _ET_multi_
#define _ET_multi_

/* base class */
#include "LAdMatrixT.h"
#include "RGSplitT2.h"
#include "InvLangevin.h"
#include "Gamma.h"

namespace Tahoe {

    class ET_multi: public RGSplitT2
    {
    public:

        /* constructor/destructor */
        ET_multi(void);

        /*Bookkeeping: initialize, update/reset internal variables and
           the beginning and end of each time step */
        virtual void PointInitialize(void);
        virtual void UpdateHistory(void); // element at a time
        virtual void ResetHistory(void);  // element at a time


        /** \name Input/Output:
         Defines the model parameters
         Reads in model parameters input file
         Overloads ParameterInterfaceT interface */
        /*@{*/
        /** information about subordinate parameter lists */
        virtual void DefineSubs(SubListT& sub_list) const;
        /** a pointer to the ParameterInterfaceT of the given
           subordinate */
        virtual ParameterInterfaceT* NewSub(const StringT& name) const;
        /** describe the parameters needed by the interface */
        virtual void DefineParameters(ParameterListT& list) const;
        /** accept parameter list */
        virtual void TakeParameterList(const ParameterListT& list);

        /*These are the main functions*/
        /*stress response*/
        /*returns the spatial tangent modulus in reduced matrix form*/
```

[2] http://sourceforge.net/projects/tahoe/

```
42          virtual const dMatrixT& c_ijkl(void);
43          /*returns the symmetric Cauchy stress tensor*/
44          virtual const dSymMatrixT& s_ij(void);
45          /*returns inverse(F_T)*/
46          virtual const dMatrixT& ThermalDeformation_Inverse(void);
47          virtual bool Need_F_last(void) const;
48
49          /*Heat generation*/
50          /** incremental heat generation */
51          virtual double IncrementalHeat(void);
52
53          /** this model does generate heat */
54          virtual bool HasIncrementalHeat(void) const { return true; };
55
56
57
58      private:
59          /*relaxation spectrum*/
60          /*returns  tau_R/tauR0, the normalized strutural relaxation time
                */
61          double StructuralRelaxationFunc(const double Temperature, const
                double Sc);
62          /*returns  tau_S/tauS0, normalized stress relaxation time*/
63          double StressRelaxationFunc(const double Temperature, const
                double smag);
64
65          double ReptationRelaxationFuncT(const double Temperature);
66          double ReptationRelaxationFuncS(const double Temperature, const
                double smag);
67
68          double capacityk(const double Temperature);
69          double capacityc(const double aTe0);
70          double falpha(const double ax);
71          double alphakf(const double lk, const double lmc);
72
73          /*Numerical integration of the evolution equations*/
74          void Compute_Tei(const dArrayT& eigs_n, const dArrayT& eigs,
                const dArrayT& eigs_tr, const dArrayT& DScDe, const dArrayT&
                 stretche,const dArrayT& stressk, const dArrayT& backstressk
                , const dArrayT& Tfk_n, dArrayT& Tfk, const dArrayT& avec,
                const dArrayT& avec_tr);
75          void Compute_a(const dArrayT& eigs_n, const dArrayT& eigs, const
                 dArrayT& eigs_tr, const dArrayT& DScDe, const dArrayT&
                stretche,const dArrayT& stressk, dArrayT& backstressk, const
                 dArrayT& Tfk_n, dArrayT& Tfk, dArrayT& avec, const dArrayT&
                 avec_tr);
76          void Compute_le(const ArrayT<dSymMatrixT>& C_vn, ArrayT<
                dSymMatrixT>& C_v, const dArrayT& Tfk_n, dArrayT& Tfk, const
                 ArrayT<dSymMatrixT>& C_Nvn, ArrayT<dSymMatrixT>& C_Nv,
                double& heat);
77          void Compute_Kneq(dMatrixT& Modulus1, dMatrixT& Modulus2);
78          int Reorder(dArrayT& EigValue,dArrayT RefValue, dMatrixT& EigVec
                , dMatrixT& RefVec);
79
80      protected:
81          /*Gamma Function*/
```

```
82          Gamma fGamma;
83
84          /*Reference Temperature*/
85          double fT0; /*Initial temperature*/
86          double fTg;         /* glass transition temperature*/
87          double fC1, fC2;    /*WLF constants*/
88
89
90          /*excess heat capacity cr-cg*/
91          double fdeltac0, fdeltac1;
92          double fcg0, fcg1;
93          /*shear moduli*/
94          double fmur;     /*the rubbery, high temperature shear modulus*/
95          double fmug;     /*the glassy, low temperature shear modulus*/
96
97          double fafrac, fBB,  fT2;
98
99          /*low temp viscous flow*/
100         double fQS;         /*activation volume for viscoplastic flow*/
101
102         /*discrete structural relaxation spectrum*/
103         int fNumR;
104         dArrayT ftimesR, fdalpha;
105         StringT fInputR;
106
107         /*discrete stress relaxation spectrum*/
108         int fNumS;
109         dArrayT ftimesS, fdmu;
110         StringT fInputS;
111
112         /*accessors for internal variables*/
113         dArrayT fTfk;
114         dArrayT fTfk_n;
115         double* fHeat;
116         double* fHeat_n;
117
118         /*workspaces*/
119         dArrayT    fl_tr;
120         dArrayT    fle;
121         dArrayT ftau;
122         dArrayT fstressk;
123
124         LAdMatrixT fKdel;
125         dArrayT fRdel;
126
127         LAdMatrixT fKAB;
128         LAdMatrixT fKAB2;
129         dArrayT fRes;
130
131         dArrayT fGA0;
132         dArrayT fGA1;
133         dArrayT fGA2;
134         dMatrixT fDAB;
135         dMatrixT fMat;
136
137         //new spectral decomps
```

```
138         SpectralDecompT fSpectralDecompb;
139         SpectralDecompT fSpectralDecompRef;
140         SpectralDecompT fSpectralDecompTest;
141 
142         /*new*/
143         dArrayT  fEigs_last, fEigs_e_last, flatenth, fGB0, fGB1, fGB2,
                fG10, fG11, fG12, fG20, fG21, fG22;
144         dMatrixT fF3D_last, fDAB2;
145         dSymMatrixT fb_last;
146         LAdMatrixT fKrTe, fKTer, fInverse1, fInverse2, fInverse3, fKdel2
                , fK1, fK2;
147 
148         /*hardening*/
149         dArrayT  fEigs_a, fEigs_a_last, fEigs_a_dev, fbackstressk,fsback
                , ftau_BACK, fResa, favec, favec_tr, fflowstressk, fzeta,
                ftau_FLOW, fGC0, fGC1, fGC2, fG10temp, fG11temp, fG12temp,
                fG20temp, fG21temp, fG22temp;
150         double fHNratio,fbfrac;
151         LAdMatrixT fKaTe, fKTea, fKaAB, fKaAB2, fKar,fKra, fK11,fK12,
                fK21,fK22, fK11temp, fK12temp, fK21temp, fK22temp, fK3, fK4,
                 fKrTe2, fKTer2, fK3temp, fK4temp, fInverse4, fInverse5;
152         dSymMatrixT fDtauDe_BACK;
153 
154         double fshift, fBN, fQSb, falphac, falpha0, fmpower;
155         dArrayT ffrac1, ffrac2, ffrac3, ffrac4, fdeltac, fvolc,
                fNstressk, fNstress, ftau_N, fepseN, fepseNp, fEigs_v;
156         ArrayT<dSymMatrixT> fC_Nv;
157         ArrayT<dSymMatrixT> fC_Nvn;
158         dMatrixT fR;
159         dSymMatrixT fU, fUv, fa;
160         dArrayT fEigs_cvn;
161         dMatrixT fFv3D;
162         dMatrixT fEigvec1, fEigvec2, fEigMatb;
163     };
164     inline bool ET_multi::Need_F_last(void) const { return true; }
165 }
166 #endif
```

APPENDIX B. FINITE ELEMENT IMPLEMENTATION OF THE CONSTITUTIVE MODEL BASED ON EFFECTIVE TEMPERATURE THEORY

Main file: ET-multi.cpp

```
#include "ET_multi.h"

#include "PotentialT.h"
#include "NeoHookean.h"
#include "ArrudaBoyce.h"

#include "ifstreamT.h"
#include "ExceptionT.h"
#include <cmath>
#include "ofstreamT.h"
#include <cstdlib>
#include "ParameterContainerT.h"

using namespace Tahoe;

const double loge = log10(exp(1.0));
const double pi = 2.0*acos(0);
const double third = 1.0/3.0;
const double small = 1.0e-7;
const double small2 = 1.0e-5;

/* ************************************************************************
 * Public
 ************************************************************************ */

/* constructors */
ET_multi::ET_multi(void):
fSpectralDecompb(3),
fSpectralDecompRef(3),
fSpectralDecompTest(3),
ParameterInterfaceT("ET_multi")
{
}

double ET_multi::StructuralRelaxationFunc(const double Temperature,
    const double Sc)
{
    /*calculate entropy at Tg*/
   double ScTg=fBB/fTg/(fdeltac0*log(fTg/fT2)+fdeltac1*(fTg-fT2));
   double coeff = (fBB/Temperature/Sc-ScTg);
   double itauR = exp(-coeff);
   return(itauR);
}

double ET_multi::ReptationRelaxationFuncT(const double Temperature)
{
   double coeff = fBN/Temperature;
   double itauR = exp(-coeff);
   return(itauR);
}

double ET_multi::ReptationRelaxationFuncS(const double Temperature,
    const double smag)
```

```
52 {
53    double itauS = 1.0;
54 
55    if (smag > kSmall)
56    {
57       itauS *= sinh(fQSb/Temperature * smag);
58       itauS *= Temperature/(fQSb*smag);
59    }
60 
61    return(itauS);
62 }
63 
64 double ET_multi::StressRelaxationFunc(const double Temperature, const
       double smag)
65 
66 {
67    double itauS = 1.0;
68 
69    if (smag > kSmall)
70    {
71       itauS *= sinh(fQS/Temperature * smag);
72       itauS *= Temperature/(fQS*smag);
73    }
74 
75    return(itauS);
76 }
77 
78 double ET_multi::capacityk(const double Temperature)
79 {
80     double fcgT=fcg0+fcg1*Temperature;
81     return fcgT;
82 }
83 
84 double  ET_multi::capacityc(const double aTe0)
85 {
86     double fdeltacTf=fdeltac0+fdeltac1*aTe0;
87     return fdeltacTf;
88 }
89 
90 double  ET_multi::falpha(const double ax)
91 {
92     double alphafunc=(falpha0-falphac)/(ax-falphac)*pow(ax, -fmpower);
93     return alphafunc;
94 }
95 
96 double  ET_multi::alphakf(const double lk, const double lmc)
97 {
98     double alphak=acos(lk/lmc);
99     return alphak;
100 }
101 
102 /* incremental heat generation */
103 double ET_multi::IncrementalHeat(void)
104 {
105    /* trust the "current" element is already loaded */
106    ElementCardT& element = CurrentElement();
```

```
107     Load(element, CurrIP());
108      return(*fHeat);
109  }
110
111  const dMatrixT& ET_multi::ThermalDeformation_Inverse(void)
112  {
113     fF_T_inv.Identity(1.0);
114     return(fF_T_inv);
115  }
116
117  /* stresses */
118  const dSymMatrixT& ET_multi::s_ij(void)
119  {
120      double Tn = Compute_Temperature();
121      /*load the viscoelastic principal stretches from state variable
           arraysand calculate NEQ part*/
122     ElementCardT& element = CurrentElement();
123     Load(element, CurrIP());
124     if (fFSMatSupport->RunState() == GlobalT::kFormRHS)
125     {
126          /*compute evolution equations for updated effective temperature
               and viscous deformation tensor*/
127        Compute_le(fC_vn, fC_v, fTfk_n, fTfk, fC_Nvn, fC_Nv, *fHeat);
128        Store(element, CurrIP());
129     }
130
131      const dMatrixT& F = F_total();
132     if (NumSD() == 2)
133     {
134        fF3D[0] = F[0];
135        fF3D[1] = F[1];
136        fF3D[2] = 0.0;
137
138        fF3D[3] = F[2];
139        fF3D[4] = F[3];
140        fF3D[5] = 0.0;
141
142        fF3D[6] = 0.0;
143        fF3D[7] = 0.0;
144        fF3D[8] = 1.0;
145     }
146     else fF3D = F;
147
148     /*calculate EQ part of the stress*/
149      fb.MultAAT(fF3D);
150     fSpectralDecompSpat.SpectralDecomp_Jacobi(fb, false);
151     fEigs = fSpectralDecompSpat.Eigenvalues();
152      double J = sqrt(fEigs.Product());
153
154      fEigs_dev = fEigs;
155     fEigs_dev *= pow(J, -2.0*third);
156     fPot[0]->DevStress(fEigs_dev, ftau_EQ);
157
158      double precoeff=0.0;
159      for(int k=0; k<fNumR; k++)
160     {
```

```
161       double phi=fdalpha[k];
162         precoeff += (fTfk[k]*phi);
163     }
164 
165     precoeff /= fT0;
166     ftau_EQ *= precoeff;
167     ftau_EQ += fPot[0]->MeanStress(J);
168    fStress3D = fSpectralDecompSpat.EigsToRank2(ftau_EQ);
169 
170    /*calc NEQ component of stress and moduli*/
171    for (int i = 0; i < fNumS; i++)
172    {
173       /*calc elastic stretch*/
174       fInverse.Inverse(fC_v[i]);
175       fbe.MultQBQT(fF3D, fInverse);
176       fSpectralDecompSpat.SpectralDecomp_Jacobi(fbe, false);
177       fEigs_e = fSpectralDecompSpat.Eigenvalues();
178 
179       double Je = sqrt(fEigs_e.Product());
180       fEigs_dev = fEigs_e;
181       fEigs_dev *= pow(Je,-2.0*third);
182 
183       fPot[1]->DevStress(fEigs_dev, ftau_NEQ);
184       ftau_NEQ *=fdmu[i];
185       fStress3D += fSpectralDecompSpat.EigsToRank2(ftau_NEQ);
186    }
187 
188    if (NumSD() == 2)
189     {
190         fStress[0] = fStress3D[0];
191         fStress[1] = fStress3D[1];
192         fStress[2] = fStress3D[5];
193     }
194     else fStress = fStress3D;
195    const dMatrixT& Ftotal = F_total();
196     fStress *= 1.0/Ftotal.Det();
197 
198    return fStress;
199 
200 }
201 
202 /* modulus */
203 const dMatrixT& ET_multi::c_ijkl(void)
204 {
205     double Tn = Compute_Temperature();
206 
207     ElementCardT& element = CurrentElement();
208     Load(element, CurrIP());
209 
210    Compute_Kneq(fDAB, fDAB2);
211 
212     const dMatrixT& F = F_total();
213    if (NumSD() == 2)
214    {
215       fF3D[0] = F[0];
216       fF3D[1] = F[1];
```

```
217       fF3D[2] = 0.0;
218
219       fF3D[3] = F[2];
220       fF3D[4] = F[3];
221       fF3D[5] = 0.0;
222
223       fF3D[6] = 0.0;
224       fF3D[7] = 0.0;
225       fF3D[8] = 1.0;
226    }
227    else fF3D = F;
228
229
230     fb.MultAAT(fF3D);
231    fSpectralDecompSpat.SpectralDecomp_Jacobi(fb, false);
232    fEigs = fSpectralDecompSpat.Eigenvalues();
233     const ArrayT<dArrayT>& eigenvectors=fSpectralDecompSpat.Eigenvectors
          ();
234
235
236     double J = sqrt(fEigs.Product());
237     fEigs_dev = fEigs;
238    fEigs_dev *= pow(J, -2.0*third);
239
240     double precoeff=0.0;
241     for(int k=0; k<fNumR; k++)
242    {
243       double phi=fdalpha[k];
244         precoeff += (fTfk[k]*phi);
245     }
246
247     precoeff /= fT0;
248
249     fPot[0]->DevStress(fEigs_dev, ftau_EQ);
250     double tau0 = ftau_EQ[0];
251     double tau1 = ftau_EQ[1];
252     double tau2 = ftau_EQ[2];
253
254     ftau_EQ *= precoeff;
255    ftau_EQ += fPot[0]->MeanStress(J);
256
257    fPot[0]->DevMod(fEigs_dev,fDtauDe_EQ);
258     fDtauDe_EQ *= precoeff;
259     fDtauDe_EQ += fPot[0]->MeanMod(J);
260
261    dSymMatrixT& Gamma = fDtauDe_EQ;
262     Gamma(0,0) -= 2.0*ftau_EQ[0];
263     Gamma(1,1) -= 2.0*ftau_EQ[1];
264     Gamma(2,2) -= 2.0*ftau_EQ[2];
265
266     for (int i = 0; i < fNumR; i++)
267     {
268         double phi=fdalpha[i];
269         double kneq10 = tau0/fT0*phi;
270         double kneq11 = tau1/fT0*phi;
271         double kneq12 = tau2/fT0*phi;
```

```
272
273         Gamma(0,0) += kneq10*fDAB(i,0);
274         Gamma(0,1) += kneq10*fDAB(i,1);
275         Gamma(0,2) += kneq10*fDAB(i,2);
276         Gamma(1,0) += kneq11*fDAB(i,0);
277         Gamma(1,1) += kneq11*fDAB(i,1);
278         Gamma(1,2) += kneq11*fDAB(i,2);
279         Gamma(2,0) += kneq12*fDAB(i,0);
280         Gamma(2,1) += kneq12*fDAB(i,1);
281         Gamma(2,2) += kneq12*fDAB(i,2);
282      }
283
284      fModulus3D = fSpectralDecompSpat.EigsToRank4(Gamma);
285    double dl, coeff;
286
287      double l0 = fEigs[0];
288      double l1 = fEigs[1];
289      double l2 = fEigs[2];
290
291    dl = l0 - l1;
292      if (fabs(dl) > kSmall)
293        coeff = (ftau_EQ[0]*l1 - ftau_EQ[1]*l0)/dl;
294      else
295        coeff = 0.5*(Gamma(0,0)-Gamma(0,1))-ftau_EQ[0];
296      MixedRank4_3D(eigenvectors[0], eigenvectors[1], fModMat);
297      fModulus3D.AddScaled(2.0*coeff, fModMat);
298
299      dl = l0 - l2;
300      if (fabs(dl) > kSmall)
301          coeff = (ftau_EQ[0]*l2 - ftau_EQ[2]*l0)/dl;
302      else
303          coeff = 0.5*(Gamma(0,0)-Gamma(0,2))-ftau_EQ[2];
304      MixedRank4_3D(eigenvectors[0], eigenvectors[2], fModMat);
305      fModulus3D.AddScaled(2.0*coeff, fModMat);
306
307      dl = l1 - l2;
308      if (fabs(dl) > kSmall)
309        coeff  = (ftau_EQ[1]*l2 - ftau_EQ[2]*l1)/dl;
310      else
311          coeff = 0.5*(Gamma(1,1)-Gamma(1,2))-ftau_EQ[1];
312      MixedRank4_3D(eigenvectors[1], eigenvectors[2], fModMat);
313      fModulus3D.AddScaled(2.0*coeff, fModMat);
314
315    for (int i = 0; i < fNumS; i++)
316    {
317
318        fInverse.Inverse(fC_v[i]);
319       fbe.MultQBQT(fF3D, fInverse);
320       fSpectralDecompSpat.SpectralDecomp_Jacobi(fbe, false);
321       fEigs_e = fSpectralDecompSpat.Eigenvalues();
322       double Je = sqrt(fEigs_e.Product());
323       fEigs_dev = fEigs_e;
324       fEigs_dev *= pow(Je,-2.0*third);
325
326       fInverse.Inverse(fC_vn[i]);
327       fb_tr.MultQBQT(fF3D, fInverse);
```

```
328         fSpectralDecompSpat.SpectralDecomp_Jacobi(fb_tr, false);
329         fEigs_tr = fSpectralDecompSpat.Eigenvalues();
330         const ArrayT<dArrayT>& eigenvectors_tr=fSpectralDecompSpat.
                Eigenvectors();
331
332         double mu = fdmu[i];
333         fPot[1]->DevStress(fEigs_dev, ftau_NEQ);
334         ftau_NEQ *= mu;
335
336         fPot[1]->DevMod(fEigs_dev, fDtauDe_NEQ);
337         fDtauDe_NEQ *=mu;
338
339         double a = 3*i;
340         fMat(0,0) = fDAB2(a,0);
341         fMat(0,1) = fDAB2(a,1);
342         fMat(0,2) = fDAB2(a,2);
343         fMat(1,0) = fDAB2(a+1,0);
344         fMat(1,1) = fDAB2(a+1,1);
345         fMat(1,2) = fDAB2(a+1,2);
346         fMat(2,0) = fDAB2(a+2,0);
347         fMat(2,1) = fDAB2(a+2,1);
348         fMat(2,2) = fDAB2(a+2,2);
349
350         fCalg(0,0) = fDtauDe_NEQ(0,0)*fMat(0,0) + fDtauDe_NEQ(0,1)*fMat
                (1,0) + fDtauDe_NEQ(0,2)*fMat(2,0);
351         fCalg(0,1) = fDtauDe_NEQ(0,0)*fMat(0,1) + fDtauDe_NEQ(0,1)*fMat
                (1,1) + fDtauDe_NEQ(0,2)*fMat(2,1);
352         fCalg(0,2) = fDtauDe_NEQ(0,0)*fMat(0,2) + fDtauDe_NEQ(0,1)*fMat
                (1,2) + fDtauDe_NEQ(0,2)*fMat(2,2);
353
354         fCalg(1,0) = fDtauDe_NEQ(1,0)*fMat(0,0) + fDtauDe_NEQ(1,1)*fMat
                (1,0) + fDtauDe_NEQ(1,2)*fMat(2,0);
355         fCalg(1,1) = fDtauDe_NEQ(1,0)*fMat(0,1) + fDtauDe_NEQ(1,1)*fMat
                (1,1) + fDtauDe_NEQ(1,2)*fMat(2,1);
356         fCalg(1,2) = fDtauDe_NEQ(1,0)*fMat(0,2) + fDtauDe_NEQ(1,1)*fMat
                (1,2) + fDtauDe_NEQ(1,2)*fMat(2,2);
357
358         fCalg(2,0) = fDtauDe_NEQ(2,0)*fMat(0,0) + fDtauDe_NEQ(2,1)*fMat
                (1,0) + fDtauDe_NEQ(2,2)*fMat(2,0);
359         fCalg(2,1) = fDtauDe_NEQ(2,0)*fMat(0,1) + fDtauDe_NEQ(2,1)*fMat
                (1,1) + fDtauDe_NEQ(2,2)*fMat(2,1);
360         fCalg(2,2) = fDtauDe_NEQ(2,0)*fMat(0,2) + fDtauDe_NEQ(2,1)*fMat
                (1,2) + fDtauDe_NEQ(2,2)*fMat(2,2);
361
362         fCalg(0,0) -= 2.0*ftau_NEQ[0];
363         fCalg(1,1) -= 2.0*ftau_NEQ[1];
364         fCalg(2,2) -= 2.0*ftau_NEQ[2];
365
366         fModulus3D += fSpectralDecompSpat.NonSymEigsToRank4(fCalg);
367
368
369         double dl_tr;
370
371         double l0_tr = fEigs_tr[0];
372         double l1_tr = fEigs_tr[1];
373         double l2_tr = fEigs_tr[2];
```

```
374
375
376         dl_tr = l0_tr - l1_tr;
377         if (fabs(dl_tr) > kSmall)
378            coeff = (ftau_NEQ[0]*l1_tr -ftau_NEQ[1]*l0_tr)/dl_tr;
379         else
380            coeff = 0.5*(fCalg(0,0)-fCalg(0,1))-ftau_NEQ[0];
381         MixedRank4_3D(eigenvectors_tr[0], eigenvectors_tr[1], fModMat);
382         fModulus3D.AddScaled(2.0*coeff, fModMat);
383
384         dl_tr = l0_tr - l2_tr;
385         if (fabs(dl_tr) > kSmall)
386            coeff =(ftau_NEQ[0]*l2_tr -ftau_NEQ[2]*l0_tr)/dl_tr;
387         else
388            coeff = 0.5*(fCalg(0,0)-fCalg(0,2))-ftau_NEQ[2];
389         MixedRank4_3D(eigenvectors_tr[0], eigenvectors_tr[2], fModMat);
390         fModulus3D.AddScaled(2.0*coeff, fModMat);
391
392         dl_tr = l1_tr - l2_tr;
393         if (fabs(dl_tr) > kSmall)
394            coeff  = (ftau_NEQ[1]*l2_tr - ftau_NEQ[2]*l1_tr)/dl_tr;
395         else
396            coeff = 0.5*(fCalg(1,1)-fCalg(1,2))-ftau_NEQ[1];
397         MixedRank4_3D(eigenvectors_tr[1], eigenvectors_tr[2], fModMat);
398         fModulus3D.AddScaled(2.0*coeff, fModMat);
399      }
400      if (NumSD() == 2)
401      {
402         fModulus[0] = fModulus3D[0];
403         fModulus[1] = fModulus3D[1];
404         fModulus[2] = fModulus3D[5];
405
406         fModulus[3] = fModulus3D[6];
407         fModulus[4] = fModulus3D[7];
408         fModulus[5] = fModulus3D[11];
409         fModulus[6] = fModulus3D[30];
410         fModulus[7] = fModulus3D[31];
411         fModulus[8] = fModulus3D[35];
412      }
413      else fModulus = fModulus3D;
414
415      const dMatrixT& Ftotal = F_total();
416      fModulus *= 1.0/Ftotal.Det();
417
418      return fModulus;
419   }
420
421   /* describe the parameters needed by the interface */
422   void ET_multi::DefineParameters(ParameterListT& list) const
423   {
424      /* inherited */
425      RGViscoelasticityT::DefineParameters(list);
426
427      /* common limit */
428      LimitT positive(0.0, LimitT::LowerInclusive);
429      LimitT zero(0.0, LimitT::Lower);
```

```
430
431     ParameterT inittemp(ParameterT::Double, "initial_temperature_T0");
432     ParameterT glasstemp(ParameterT::Double, "glass_trans_temp_Tg");
433     ParameterT C1(ParameterT::Double, "WLF_C1");
434     ParameterT C2(ParameterT::Double, "WLF_C2");
435
436     ParameterT afrac(ParameterT::Double, "fraction_a");
437     ParameterT BB(ParameterT::Double, "B");
438     ParameterT deltac0(ParameterT::Double, "delta_c0");
439     ParameterT deltac1(ParameterT::Double, "delta_c1");
440     ParameterT cg0(ParameterT::Double, "c_g0");
441     ParameterT cg1(ParameterT::Double, "c_g1");
442     ParameterT T2(ParameterT::Double, "T2");
443
444     ParameterT bfrac(ParameterT::Double, "fraction_b");
445     ParameterT HNratio(ParameterT::Double, "HardeningMod_NeqMod");
446     ParameterT shift(ParameterT::Double, "rep_relaxation_time_shift");
447
448     ParameterT BN(ParameterT::Double, "temp_coeff_for_rep");
449     ParameterT QSb(ParameterT::Double, "activation_vol_for_rep");
450     ParameterT alphac(ParameterT::Double, "alphac");
451     ParameterT alpha0(ParameterT::Double, "alpha0");
452     ParameterT mpower(ParameterT::Double, "mpower");
453
454     inittemp.AddLimit(positive);
455     glasstemp.AddLimit(positive);
456     C1.AddLimit(positive);
457     C2.AddLimit(positive);
458
459     afrac.AddLimit(positive);
460     BB.AddLimit(positive);
461     T2.AddLimit(positive);
462
463     bfrac.AddLimit(positive);
464     shift.AddLimit(positive);
465     HNratio.AddLimit(positive);
466
467     BN.AddLimit(positive);
468     QSb.AddLimit(positive);
469     alphac.AddLimit(positive);
470     alpha0.AddLimit(positive);
471     mpower.AddLimit(positive);
472
473     list.AddParameter(inittemp);
474     list.AddParameter(glasstemp);
475     list.AddParameter(C1);
476     list.AddParameter(C2);
477
478     list.AddParameter(afrac);
479     list.AddParameter(BB);
480     list.AddParameter(deltac0);
481     list.AddParameter(deltac1);
482     list.AddParameter(cg0);
483     list.AddParameter(cg1);
484     list.AddParameter(T2);
485
```

```
486     list.AddParameter(bfrac);
487     list.AddParameter(shift);
488     list.AddParameter(HNratio);
489 
490     list.AddParameter(BN);
491     list.AddParameter(QSb);
492     list.AddParameter(alphac);
493     list.AddParameter(alpha0);
494     list.AddParameter(mpower);
495 }
496 
497 /* information about subordinate parameter lists */
498 void ET_multi::DefineSubs(SubListT& sub_list) const
499 {
500    /* inherited */
501    FSSolidMatT::DefineSubs(sub_list);
502 
503    sub_list.AddSub("ET_multi_eq_potential", ParameterListT::Once);
504    sub_list.AddSub("ET_multi_neq_potential", ParameterListT::Once);
505     sub_list.AddSub("ET_multi_backstress_potential", ParameterListT::
          Once);
506 
507    /* choice of viscosity */
508    sub_list.AddSub("ET_multi_structural_spectrum", ParameterListT::Once)
          ;
509    sub_list.AddSub("ET_multi_stress_spectrum", ParameterListT::Once);
510 }
511 
512 
513 /* a pointer to the ParameterInterfaceT of the given subordinate */
514 ParameterInterfaceT* ET_multi::NewSub(const StringT& name) const
515 {
516     LimitT zero(0.0, LimitT::Lower);
517    LimitT one(1.0, LimitT::Upper);
518    LimitT two(2, LimitT::LowerInclusive);
519    LimitT positive(0.0, LimitT::LowerInclusive);
520 
521    PotentialT* pot = NULL;
522    if (name == "neo-hookean")
523       pot = new NeoHookean;
524    else if (name == "arruda-boyce")
525       pot = new ArrudaBoyce;
526    if (pot)
527       return pot;
528 
529    /* inherited */
530    ParameterInterfaceT* sub = RGViscoelasticityT::NewSub(name);
531    if (sub)
532    {
533       return sub;
534    }
535    else if (name == "ET_multi_eq_potential")
536    {
537       ParameterContainerT* choice = new ParameterContainerT(name);
538       choice->SetListOrder(ParameterListT::Choice);
539       choice->SetSubSource(this);
```

```
540         choice->SetDescription("temperature normalized network stiffness")
                ;
541
542         /* choice of parameters */
543         choice->AddSub("arruda-boyce");
544         choice->AddSub("neo-hookean");
545         return(choice);
546      }
547      else if (name == "ET_multi_neq_potential")
548      {
549         ParameterContainerT* choice = new ParameterContainerT(name);
550         choice->SetListOrder(ParameterListT::Choice);
551         choice->SetSubSource(this);
552
553         /* choice of parameters */
554         choice->AddSub("neo-hookean");
555         return(choice);
556      }
557       else if (name == "ET_multi_backstress_potential")
558      {
559         ParameterContainerT* choice = new ParameterContainerT(name);
560         choice->SetListOrder(ParameterListT::Choice);
561         choice->SetSubSource(this);
562
563         /* choice of parameters */
564         choice->AddSub("arruda-boyce");
565         return(choice);
566      }
567      else if (name == "ET_multi_structural_spectrum")
568      {
569         ParameterContainerT* tauR = new ParameterContainerT(name);
570
571         ParameterT file(ParameterT::String, "user_input_file");
572
573         tauR->AddParameter(file);
574
575         return(tauR);
576      }
577      else if (name == "ET_multi_stress_spectrum")
578      {
579         ParameterContainerT* tauS = new ParameterContainerT(name);
580
581         ParameterT file(ParameterT::String, "user_input_file");
582         tauS->AddParameter(file);
583
584         ParameterT A(ParameterT::Double, "activation_volume");
585
586         A.AddLimit(zero);
587
588         tauS->AddParameter(A);
589
590         return(tauS);
591      }
592   }
593
594   void ET_multi::TakeParameterList(const ParameterListT& list)
```

```
595 {
596     const char caller[] = "ET_multi::TakeParameterList";
597     /* inherited */
598     RGViscoelasticityT::TakeParameterList(list);
599
600
601     fT0 = list.GetParameter("initial_temperature_T0");
602     fTg = list.GetParameter("glass_trans_temp_Tg");
603     fC1 = list.GetParameter("WLF_C1");
604     fC2 = list.GetParameter("WLF_C2");
605
606     fafrac = list.GetParameter("fraction_a");
607     fBB = list.GetParameter("B");
608     fdeltac0 = list.GetParameter("delta_c0");
609     fdeltac1 = list.GetParameter("delta_c1");
610     fcg0 = list.GetParameter("c_g0");
611     fcg1 = list.GetParameter("c_g1");
612     fT2 = list.GetParameter("T2");
613
614     fbfrac = list.GetParameter("fraction_b");
615     fshift = list.GetParameter("rep_relaxation_time_shift");
616     fHNratio = list.GetParameter("HardeningMod_NeqMod");
617
618     fBN = list.GetParameter("temp_coeff_for_rep");
619     fQSb = list.GetParameter("activation_vol_for_rep");
620     falphac = list.GetParameter("alphac");
621     falpha0 = list.GetParameter("alpha0");
622     fmpower = list.GetParameter("mpower");
623
624   fPot.Dimension(3);
625
626   const ParameterListT& eq_pot = list.GetListChoice(*this, "
        ET_multi_eq_potential");
627   if(eq_pot.Name() == "arruda-boyce")
628     fPot[0] = new ArrudaBoyce;
629   else if(eq_pot.Name() == "neo-hookean")
630     fPot[0] = new NeoHookean;
631   else
632     ExceptionT::GeneralFail(caller, "no such potential");
633   if (!fPot[0]) ExceptionT::GeneralFail(caller, "could not construct
        \"%s\"", eq_pot.Name().Pointer());
634   fPot[0]->TakeParameterList(eq_pot);
635   /*set the rubbery modulus*/
636   fmur = fPot[0]->GetMu();
637
638   const ParameterListT& neq_pot = list.GetListChoice(*this, "
        ET_multi_neq_potential");
639   if(neq_pot.Name() == "neo-hookean")
640     fPot[1] = new NeoHookean;
641   else
642     ExceptionT::GeneralFail(caller, "no such potential");
643   if (!fPot[1]) ExceptionT::GeneralFail(caller, "could not construct
        \"%s\"", neq_pot.Name().Pointer());
644   fPot[1]->TakeParameterList(neq_pot);
645   /*set the glassy modulus*/
646   fmug = fmur + fPot[1]->GetMu();
```

```
647     /*normalize the neq potential function by setting the shear modulus
            to one*/
648     fPot[1]->SetMu(1.0);
649
650      const ParameterListT& backstress_pot = list.GetListChoice(*this, "
            ET_multi_backstress_potential");
651     if(backstress_pot.Name() == "arruda-boyce")
652        fPot[2] = new ArrudaBoyce;
653     else
654        ExceptionT::GeneralFail(caller, "no such potential");
655     if (!fPot[2]) ExceptionT::GeneralFail(caller, "could not construct
            \"%s\"", backstress_pot.Name().Pointer());
656     fPot[2]->TakeParameterList(backstress_pot);
657     fPot[2]->SetMu(1.0);
658
659      const ParameterListT* tauR = list.List("ET_multi_structural_spectrum
            ");
660      if (tauR)
661      {
662          fInputR = tauR->GetParameter("user_input_file");
663      }
664
665      const ParameterListT* tauS = list.List("ET_multi_stress_spectrum");
666      if (tauS)
667      {
668          fInputS = tauS->GetParameter("user_input_file");
669
670          fQS = tauS->GetParameter("activation_volume");
671      }
672
673     /*read in structural relaxation spectrum*/
674     ifstreamT inR;
675     inR.open(fInputR);
676     if(!inR.good())
677        ExceptionT::DatabaseFail(caller,
678                                 "could not open file \"%s\"",fInputR.
                                        Pointer());
679     inR >> fNumR;
680     ftimesR.Dimension(fNumR);
681     fdalpha.Dimension(fNumR);
682     for (int i=0; i<fNumR && inR.good(); i++)
683     {
684        inR >> ftimesR[i];
685        inR >> fdalpha[i];
686     }
687     inR.close();
688
689     /*read in stress relaxation spectrum*/
690     ifstreamT inS;
691     inS.open(fInputS);
692     if(!inS.good())
693        ExceptionT::DatabaseFail(caller,
694                                 "could not open file \"%s\"",fInputS.
                                        Pointer());
695     inS >> fNumS;
696     ftimesS.Dimension(fNumS);
```

```
697     fdmu.Dimension(fNumS);
698     for (int i=0; i<fNumS && inS.good(); i++)
699     {
700        inS >> ftimesS[i];
701        inS >> fdmu[i];
702     }
703     inS.close();
704
705     /*Dimension workspace*/
706     /*Dimension state variable accessors*/
707     fC_v.Dimension(fNumS);
708     fC_vn.Dimension(fNumS);
709      fC_Nv.Dimension(fNumS);
710      fC_Nvn.Dimension(fNumS);
711     fl_tr.Dimension(fNumS*3);
712     fle.Dimension(fNumS*3);
713      favec_tr.Dimension(fNumS*3);
714      favec.Dimension(fNumS*3);
715     fstressk.Dimension(fNumS*3);
716      fbackstressk.Dimension(fNumS*3);
717      fflowstressk.Dimension(fNumS*3);
718
719     int nsd = NumSD();
720     int ndof = 3;
721     int numstress = dSymMatrixT::NumValues(ndof);
722
723      /*Set pointer to internal state variables*/
724     fnstatev = 0;
725     fnstatev += numstress*fNumS;    /*current C_v*/
726     fnstatev += numstress*fNumS;    /*last C_vn*/
727      fnstatev += numstress*fNumS;    /*current CN_v*/
728      fnstatev += numstress*fNumS;    /*last CN_vn*/
729     fnstatev += fNumR;          /*current  Tfk*/
730     fnstatev += fNumR;          /*last Tfk*/
731      fnstatev ++;                 /*fHeat incremental heat calculated from
            dotCv and dotTfk*/
732      fnstatev ++;                 /*fHeat_n last incremental heat
           calculated from dotCv and dotTfk*/
733
734     fstatev.Dimension(fnstatev);
735     double* pstatev = fstatev.Pointer();
736
737     /* assign pointers to current and last blocks of state variable array
          */
738     for (int i = 0; i < fNumS; i++)
739     {
740        fC_v[i].Set(ndof, pstatev);
741        pstatev += numstress;
742        fC_vn[i].Set(ndof, pstatev);
743        pstatev += numstress;
744     }
745      for (int i = 0; i < fNumS; i++)
746      {
747          fC_Nv[i].Set(ndof, pstatev);
748          pstatev += numstress;
749          fC_Nvn[i].Set(ndof, pstatev);
```

```
750         pstatev += numstress;
751      }
752
753     fTfk.Alias(fNumR,pstatev);
754     pstatev += fNumR;
755     fTfk_n.Alias(fNumR,pstatev);
756     pstatev += fNumR;
757
758     fHeat = pstatev;
759     pstatev++;
760      fHeat_n = pstatev;
761      pstatev++;
762
763     fF_M.Dimension(nsd);
764     fF_T_inv.Dimension(nsd);
765      fF_T_inv.Identity(1.0);
766
767     fF3D.Dimension(ndof);
768     fInverse.Dimension(ndof);
769      fF3D_last.Dimension(ndof);
770      fFv3D.Dimension(ndof);
771      fU.Dimension(ndof);
772      fR.Dimension(ndof);
773      fa.Dimension(ndof);
774      fEigvec1.Dimension(ndof);
775      fEigvec2.Dimension(ndof);
776      fEigMatb.Dimension(ndof);
777
778     fb.Dimension(ndof);
779     fbe.Dimension(ndof);
780     fb_tr.Dimension(ndof);
781      fb_last.Dimension(ndof);
782
783     fEigs_dev.Dimension(ndof);
784     fEigs.Dimension(ndof);
785     fEigs_e.Dimension(ndof);
786     fEigs_tr.Dimension(ndof);
787     fEigs_last.Dimension(ndof);
788     fEigs_e_last.Dimension(ndof);
789      fEigs_a.Dimension(ndof);
790      fEigs_a_last.Dimension(ndof);
791      fEigs_a_dev.Dimension(ndof);
792
793
794     ftau_EQ.Dimension(ndof);
795     ftau_NEQ.Dimension(ndof);
796     ftau.Dimension(ndof);
797      ftau_BACK.Dimension(ndof);
798      fsback.Dimension(ndof);
799      ftau_FLOW.Dimension(ndof);
800      fzeta.Dimension(ndof);
801
802     fStress.Dimension(NumSD());
803     fStress3D.Dimension(ndof);
804
805     fDtauDe_EQ.Dimension(ndof);
```

```
806     fDtauDe_NEQ.Dimension(ndof);
807     fDtauDe_BACK.Dimension(ndof);
808
809     fKdel.Dimension(fNumR);
810     fRdel.Dimension(fNumR);
811       flatenth.Dimension(fNumR);
812       fGB0.Dimension(fNumR);
813       fGB1.Dimension(fNumR);
814       fGB2.Dimension(fNumR);
815       fG10.Dimension(fNumR);
816       fG11.Dimension(fNumR);
817       fG12.Dimension(fNumR);
818       fG10temp.Dimension(fNumR);
819       fG11temp.Dimension(fNumR);
820       fG12temp.Dimension(fNumR);
821
822     fKAB.Dimension(fNumS*3);
823     fKAB2.Dimension(fNumS*3);
824       fKaAB.Dimension(fNumS*3);
825     fKaAB2.Dimension(fNumS*3);
826     fRes.Dimension(fNumS*3);
827       fResa.Dimension(fNumS*3);
828
829       fGA0.Dimension(fNumS*3);
830       fGA1.Dimension(fNumS*3);
831       fGA2.Dimension(fNumS*3);
832       fG20.Dimension(fNumS*3);
833       fG21.Dimension(fNumS*3);
834       fG22.Dimension(fNumS*3);
835       fGC0.Dimension(fNumS*3);
836       fGC1.Dimension(fNumS*3);
837       fGC2.Dimension(fNumS*3);
838       fG20temp.Dimension(fNumS*3);
839       fG21temp.Dimension(fNumS*3);
840       fG22temp.Dimension(fNumS*3);
841
842     fCalg.Dimension(3);
843     fMat.Dimension(3);
844     fDAB.Dimension(fNumR,3);
845     fDAB2.Dimension(fNumS*3,3);
846
847       fKrTe.Dimension(3*fNumS,fNumR);
848       fKTer.Dimension(fNumR,3*fNumS);
849       fKaTe.Dimension(3*fNumS,fNumR);
850       fKTea.Dimension(fNumR,3*fNumS);
851       fKar.Dimension(3*fNumS,3*fNumS);
852       fKra.Dimension(3*fNumS,3*fNumS);
853       fInverse1.Dimension(fNumR,3*fNumS);
854       fInverse2.Dimension(fNumR,3*fNumS);
855       fInverse3.Dimension(3*fNumS,3*fNumS);
856       fInverse4.Dimension(fNumR,3*fNumS);
857       fInverse5.Dimension(3*fNumS,fNumR);
858       fKdel2.Dimension(fNumR,fNumR);
859       fK1.Dimension(fNumR,fNumR);
860       fK2.Dimension(3*fNumS,3*fNumS);
861       fK3.Dimension(fNumR,fNumR);
```

```
862     fK4.Dimension(3*fNumS,3*fNumS);
863     fK3temp.Dimension(fNumR,fNumR);
864     fK4temp.Dimension(3*fNumS,3*fNumS);
865     fKrTe2.Dimension(3*fNumS,fNumR);
866     fKTer2.Dimension(fNumR,3*fNumS);
867     fK11.Dimension(3*fNumS,3*fNumS);
868     fK11temp.Dimension(3*fNumS,3*fNumS);
869     fK12.Dimension(3*fNumS,3*fNumS);
870     fK12temp.Dimension(3*fNumS,3*fNumS);
871     fK21.Dimension(fNumR,3*fNumS);
872     fK21temp.Dimension(fNumR,3*fNumS);
873     fK22.Dimension(fNumR,fNumR);
874     fK22temp.Dimension(fNumR,fNumR);
875
876     fModulus3D.Dimension(dSymMatrixT::NumValues(ndof));
877     fModMat.Dimension(dSymMatrixT::NumValues(ndof));
878     fModulus.Dimension(dSymMatrixT::NumValues(NumSD()));
879
880     ffrac1.Dimension(fNumR);
881     ffrac2.Dimension(fNumR);
882     ffrac3.Dimension(fNumR);
883     ffrac4.Dimension(fNumR);
884     fdeltac.Dimension(fNumR);
885     fNstressk.Dimension(fNumS*3);
886     fNstress.Dimension(ndof);
887     ftau_N.Dimension(ndof);
888     fepseN.Dimension(ndof);
889     fepseNp.Dimension(ndof);
890     fEigs_v.Dimension(ndof);
891     fEigs_cvn.Dimension(ndof);
892
893 }
894
895 /*initializes history variable */
896 void  ET_multi::PointInitialize(void)
897 {
898    /* allocate element storage */
899     const double Temp = Compute_Temperature();
900    ElementCardT& element = CurrentElement();
901    if (CurrIP() == 0)
902    {
903       ElementCardT& element = CurrentElement();
904       element.Dimension(0, fnstatev*NumIP());
905
906       /* initialize internal variables to identity*/
907       for (int ip = 0; ip < NumIP(); ip++)
908       {
909             /* load state variables */
910             Load(element, ip);
911             for (int k = 0; k < fNumS; k++)
912             {
913             fC_v[k].Identity();
914             fC_vn[k].Identity();
915             }
916             for (int k = 0; k < fNumS; k++)
917             {
```

```
                fC_Nv[k].Identity();
                fC_Nvn[k].Identity();
            }
            fTfk = Temp;
            fTfk_n = Temp;
            *fHeat = 0.0;
            *fHeat_n = 0.0;

            /* write to storage */
            Store(element, ip);
        }
    }
}

void ET_multi::UpdateHistory(void)
{
    /* current element */
    ElementCardT& element = CurrentElement();
    for (int ip = 0; ip < NumIP(); ip++)
    {
        /* load state variables */
        Load(element, ip);

        /* assign "current" to "last" */
        for (int k = 0; k < fNumS; k++)
          {
           fC_vn[k] = fC_v[k];
          }
          for (int k = 0; k < fNumS; k++)
          {
              fC_Nvn[k] = fC_Nv[k];
          }
        fTfk_n = fTfk;
          *fHeat_n = *fHeat;

        /* write to storage */
        Store(element, ip);
    }
}

void ET_multi::ResetHistory(void)
{
     /* current element */
    ElementCardT& element = CurrentElement();
    for (int ip = 0; ip < NumIP(); ip++)
    {
        /* load state variables*/
        Load(element, ip);

        /* assign "last" to "current" */
        for (int k = 0; k < fNumS; k++)
          {
           fC_v[k] = fC_vn[k];
          }
          for (int k = 0; k < fNumS; k++)
          {
```

```
974                 fC_Nv[k] = fC_Nvn[k];
975               }
976             fTfk = fTfk_n;
977               *fHeat = *fHeat_n;
978
979             /* write to storage */
980             Store(element, ip);
981         }
982 }
983
984 void ET_multi::Compute_Tei(const dArrayT& eigs_n, const dArrayT& eigs,
        const dArrayT& eigs_tr, const dArrayT& DScDe, const dArrayT&
        stretche,const dArrayT& stressk, const dArrayT& backstressk, const
        dArrayT& Tfk_n, dArrayT& Tfk, const dArrayT& avec, const dArrayT&
        avec_tr)
985 {
986     double ctol = 1.0e-9;
987    /*time step*/
988    double dt = fFSMatSupport->TimeStep();
989    /*current temperature*/
990    double Tn = Compute_Temperature();
991     const double Tp = Compute_Temperature_last();
992     double energy=0.0;
993     double plasticw = 0.0;
994     double Tfn = 0.0;
995     double Tfp = 0.0;
996
997     double l0 = eigs[0];
998     double l1 = eigs[1];
999     double l2 = eigs[2];
1000    double J = sqrt(l0*l1*l2);
1001    double iJ = 1.0/J;
1002
1003    double l0_last = eigs_n[0];
1004    double l1_last = eigs_n[1];
1005    double l2_last = eigs_n[2];
1006    double Jp = sqrt(l0_last*l1_last*l2_last);
1007
1008    double depsilon0=(0.5*log(l0)-0.5*log(l0_last));
1009    double depsilon1=(0.5*log(l1)-0.5*log(l1_last));
1010    double depsilon2=(0.5*log(l2)-0.5*log(l2_last));
1011
1012     double prefactor = 1.0;
1013   double ScTn = 0.0;
1014    double prefactor2 = 1.0;
1015   for(int k=0; k<fNumR; k++)
1016   {
1017      double phi=fdalpha[k];
1018        fdeltac[k]=capacityc(Tfk[k]);
1019        ffrac1[k]=1.0/fdeltac[k];
1020        ffrac2[k]=fafrac/fdeltac[k];
1021        ffrac3[k]=fbfrac/fdeltac[k];
1022        Tfn += (Tfk[k]*phi);
1023        Tfp += (Tfk_n[k]*phi);
1024        ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-fT2);
1025        prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
```

```
1026        prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
1027        flatenth[k] =  Tfk[k]/fT0*(DScDe[0]*depsilon0+DScDe[1]*depsilon1
                +DScDe[2]*depsilon2);
1028    }
1029
1030    /*calculate smag and plastic power*/
1031    const double* pstress = stressk.Pointer();
1032    const double* ple = stretche.Pointer();
1033    const double* pltr = eigs_tr.Pointer();
1034    const double* pbackstress = backstressk.Pointer();
1035    const double* pavectr = avec_tr.Pointer();
1036    const double* pavec = avec.Pointer();
1037
1038    plasticw =0.0;
1039    double backw=0.0;
1040    double tau0 = 0.0;
1041    double tau1 = 0.0;
1042    double tau2 = 0.0;
1043   for(int k=0; k<fNumS; k++)
1044   {
1045        double tauk0 = *pstress++;
1046        double tauk1 = *pstress++;
1047        double tauk2 = *pstress++;
1048
1049        double sbackk0 = *pbackstress++;
1050        double sbackk1 = *pbackstress++;
1051        double sbackk2 = *pbackstress++;
1052
1053        tau0 += tauk0;
1054        tau1 += tauk1;
1055        tau2 += tauk2;
1056
1057        double epse0 = 0.5*log(*ple++);
1058        double epse1 = 0.5*log(*ple++);
1059        double epse2 = 0.5*log(*ple++);
1060
1061        double epstr0 = 0.5*log(*pltr++);
1062        double epstr1 = 0.5*log(*pltr++);
1063        double epstr2 = 0.5*log(*pltr++);
1064
1065        double epse0N = 0.5*log(*pavec++);
1066        double epse1N = 0.5*log(*pavec++);
1067        double epse2N = 0.5*log(*pavec++);
1068
1069        double epstr0N = 0.5*log(*pavectr++);
1070        double epstr1N = 0.5*log(*pavectr++);
1071        double epstr2N = 0.5*log(*pavectr++);
1072
1073        double plasticwk = -(tauk0*(epse0-epstr0)+ tauk1*(epse1-epstr1)
                + tauk2*(epse2-epstr2));
1074        plasticw += plasticwk;
1075
1076        double backwk = sbackk0*(epse0N-epstr0N)+ sbackk1*(epse1N-
                epstr1N) + sbackk2*(epse2N-epstr2N);
1077        backw += backwk;
1078   }
```

```
1079
1080    fEigs_dev = eigs;
1081    fEigs_dev *= pow(J, -2.0*third);
1082     energy = fPot[0]->Energy(fEigs_dev, J);
1083     energy -= fPot[0]->MeanEnergy(J);
1084     ScTn -= energy/fT0;
1085     double itaubar = StructuralRelaxationFunc(Tn, ScTn);
1086
1087    double temp = 0.0;
1088     for(int k=0; k<fNumR; k++)
1089    {
1090       double itauRk = itaubar/ftimesR[k];
1091         /*residual*/
1092       fRdel[k] = Tfk[k] - Tfk_n[k] + dt*itauRk*(Tfk[k]-Tn)-ffrac1[k]*
              flatenth[k]-ffrac2[k]*plasticw+ffrac3[k]*backw;
1093       temp += fRdel[k]*fRdel[k];
1094    }
1095
1096    double tol = sqrt(temp);
1097     int maxiter=20;
1098     int iter = 0;
1099
1100     while (tol > ctol  && iter < maxiter)
1101    {
1102
1103         iter++;
1104       fKdel = 0.0;
1105       for(int k=0; k<fNumR; k++)
1106       {
1107          /*stiffness matrix*/
1108          double itauRk = itaubar/ftimesR[k];
1109             fKdel(k,k) = 1.0 + dt*itauRk-ffrac1[k]*flatenth[k]/Tfk[k]+
                 fafrac*ffrac1[k]*ffrac1[k]*fdeltac1*plasticw+ffrac1[k]*
                 ffrac1[k]*fdeltac1*flatenth[k]-fbfrac*ffrac1[k]*ffrac1[k
                 ]*fdeltac1*backw;
1110          for (int l=0; l<fNumR; l++)
1111             {
1112                 double phi=fdalpha[l];
1113                 double DtaubarDTfk = -fBB/Tn/ScTn/ScTn*(phi*fdeltac0/Tfk
                       [l]+fdeltac1*phi);
1114             fKdel(k,l) += -dt*itauRk*(Tfk[k]-Tn)*DtaubarDTfk;
                           }
1115       }
1116
1117       /*Solve for update*/
1118       fKdel.LinearSolve(fRdel);
1119       Tfk -= fRdel;
1120
1121       /*calculate residual*/
1122        prefactor = 1.0;
1123        ScTn = 0.0;
1124        prefactor2 = 1.0;
1125        Tfn =0.0;
1126         for(int k=0; k<fNumR; k++)
1127         {
1128             double phi=fdalpha[k];
```

```
1129                fdeltac[k]=capacityc(Tfk[k]);
1130                ffrac1[k]=1.0/fdeltac[k];
1131                ffrac2[k]=fafrac/fdeltac[k];
1132                ffrac3[k]=fbfrac/fdeltac[k];
1133                Tfn += (Tfk[k]*phi);
1134                ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-
                        fT2);
1135                prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
1136                prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
1137                flatenth[k] =  Tfk[k]/fT0*(DScDe[0]*depsilon0+DScDe[1]*
                        depsilon1+DScDe[2]*depsilon2);
1138            }
1139
1140             fEigs_dev = eigs;
1141            fEigs_dev *= pow(J, -2.0*third);
1142            energy = fPot[0]->Energy(fEigs_dev, J);
1143            energy -= fPot[0]->MeanEnergy(J);
1144            ScTn -= energy/fT0;
1145
1146            itaubar = StructuralRelaxationFunc(Tn, ScTn);
1147            double temp = 0.0;
1148            for(int k=0; k<fNumR; k++)
1149            {
1150                double itauRk = itaubar/ftimesR[k];
1151                /*residual*/
1152                fRdel[k] = Tfk[k] - Tfk_n[k] + dt*itauRk*(Tfk[k]-Tn)-ffrac1[
                        k]*flatenth[k]-ffrac2[k]*plasticw+ffrac3[k]*backw;
1153                temp += fRdel[k]*fRdel[k];
1154            }
1155
1156            tol = sqrt(temp);
1157
1158        }
1159
1160         if (iter >= maxiter)
1161        {
1162           cout<<"\n Number of iteration exceeds maximum. tol:  "<<tol;
1163           cout<< "\nelem: "<<CurrElementNumber()<<"\nIP: "<<CurrIP();
1164           ExceptionT::GeneralFail("ET_multi::Compute_Tei", "number of
                    iteration exceeds maximum");
1165        }
1166    }
1167
1168    void ET_multi::Compute_Kneq(dMatrixT& Modulus1, dMatrixT& Modulus2)
1169    {
1170        const double dt = fFSMatSupport->TimeStep();
1171
1172       /*temperature and temperature step*/
1173        const double Tn = Compute_Temperature();
1174        const double Tp = Compute_Temperature_last();
1175
1176        const dMatrixT& F_last = F_total_last();
1177        if (NumSD() == 2)
1178       {
1179           fF3D_last[0] = F_last[0];
1180           fF3D_last[1] = F_last[1];
```

```
1181        fF3D_last[2] = 0.0;
1182
1183        fF3D_last[3] = F_last[2];
1184        fF3D_last[4] = F_last[3];
1185        fF3D_last[5] = 0.0;
1186
1187        fF3D_last[6] = 0.0;
1188        fF3D_last[7] = 0.0;
1189        fF3D_last[8] = 1.0;
1190     }
1191     else fF3D_last = F_last;
1192
1193      fb_last.MultAAT(fF3D_last);
1194     fSpectralDecompSpat.SpectralDecomp_Jacobi(fb_last, false);
1195     fEigs_last = fSpectralDecompSpat.Eigenvalues();
1196
1197      double l0_last = fEigs_last[0];
1198      double l1_last = fEigs_last[1];
1199      double l2_last = fEigs_last[2];
1200      double Jp = sqrt(l0_last*l1_last*l2_last);
1201
1202      const dMatrixT& F = F_total();
1203     if (NumSD() == 2)
1204     {
1205        fF3D[0] = F[0];
1206        fF3D[1] = F[1];
1207        fF3D[2] = 0.0;
1208
1209        fF3D[3] = F[2];
1210        fF3D[4] = F[3];
1211        fF3D[5] = 0.0;
1212
1213        fF3D[6] = 0.0;
1214        fF3D[7] = 0.0;
1215        fF3D[8] = 1.0;
1216     }
1217     else fF3D = F;
1218
1219      fb.MultAAT(fF3D);
1220     fSpectralDecompSpat.SpectralDecomp_Jacobi(fb, false);
1221     fEigs = fSpectralDecompSpat.Eigenvalues();
1222      double J = sqrt(fEigs.Product());
1223     double iJ = 1.0/J;
1224
1225      fEigs_dev = fEigs;
1226     fEigs_dev *= pow(J, -2.0*third);
1227     fPot[0]->DevStress(fEigs_dev, ftau_EQ);
1228
1229      double l0 = fEigs[0];
1230      double l1 = fEigs[1];
1231      double l2 = fEigs[2];
1232
1233      double l0f = ftau_EQ[0]/fT0;
1234      double l1f = ftau_EQ[1]/fT0;
1235      double l2f = ftau_EQ[2]/fT0;
1236
```

```
1237      fPot[0]->DevMod(fEigs_dev,fDtauDe_EQ);
1238
1239      double depsilon0=(0.5*log(l0)-0.5*log(l0_last));
1240      double depsilon1=(0.5*log(l1)-0.5*log(l1_last));
1241      double depsilon2=(0.5*log(l2)-0.5*log(l2_last));
1242
1243      double ScTn = 0.0;
1244      double prefactor = 1.0;
1245      double prefactor2 = 1.0;
1246      double Tfn = 0.0;
1247      double Tfp = 0.0;
1248
1249    for(int k=0; k<fNumR; k++)
1250    {
1251       double phi=fdalpha[k];
1252         fdeltac[k]=capacityc(fTfk[k]);
1253         ffrac1[k]=1.0/fdeltac[k];
1254         ffrac2[k]=fafrac/fdeltac[k];
1255         ffrac3[k]=fbfrac/fdeltac[k];
1256         Tfn += (fTfk[k]*phi);
1257         Tfp += (fTfk_n[k]*phi);
1258         ScTn += fdeltac0*phi*log(fTfk[k]/fT2)+fdeltac1*phi*(fTfk[k]-fT2)
                ;
1259         prefactor += phi*fafrac*(Tn/fTfk[k]-1.0);
1260         prefactor2 += phi*fbfrac*(Tn/fTfk[k]-1.0);
1261         flatenth[k] =  fTfk[k]/fT0*(ftau_EQ[0]*depsilon0+ftau_EQ[1]*
                depsilon1+ftau_EQ[2]*depsilon2);
1262      }
1263
1264    fEigs_dev = fEigs;
1265    fEigs_dev *= pow(J, -2.0*third);
1266     double energy = fPot[0]->Energy(fEigs_dev, J);
1267     energy -= fPot[0]->MeanEnergy(J);
1268     ScTn -= energy/fT0;
1269
1270     double* pltr = fl_tr.Pointer();
1271    double* ple = fle.Pointer();
1272     double* pstress = fstressk.Pointer();
1273     double* pbackstress = fbackstressk.Pointer();
1274     double* pflowstress = fflowstressk.Pointer();
1275     double* pavectr = favec_tr.Pointer();
1276     double* pavec = favec.Pointer();
1277     double* pNstress = fNstressk.Pointer();
1278
1279     ftau = 0.0;
1280     fsback = 0.0;
1281     fzeta=0.0;
1282     fNstress=0.0;
1283     double backw=0.0;
1284     double plasticw = 0.0;
1285    /*calculate trial state*/
1286    for (int i = 0; i < fNumS; i++)
1287    {
1288         fInverse.Inverse(fC_vn[i]);
1289         fb_tr.MultQBQT(fF3D, fInverse);
1290         fSpectralDecompSpat.SpectralDecomp_Jacobi(fb_tr, false);
```

```
1291          fEigs_tr = fSpectralDecompSpat.Eigenvalues();
1292          *pltr++ = fEigs_tr[0];
1293          *pltr++ = fEigs_tr[1];
1294          *pltr++ = fEigs_tr[2];
1295          double epstr0 = 0.5*log(fEigs_tr[0]);
1296          double epstr1 = 0.5*log(fEigs_tr[1]);
1297          double epstr2 = 0.5*log(fEigs_tr[2]);
1298
1299        /*initial condition be = btr*/
1300          fInverse.Inverse(fC_v[i]);
1301          fbe.MultQBQT(fF3D, fInverse);
1302          fSpectralDecompSpat.SpectralDecomp_Jacobi(fbe, false);
1303
1304          fEigvec2=fSpectralDecompSpat.Eigenmatrix();
1305
1306          fEigs_e = fSpectralDecompSpat.Eigenvalues();
1307          *ple++ = fEigs_e[0];
1308          *ple++ = fEigs_e[1];
1309          *ple++ = fEigs_e[2];
1310          double epse0 = 0.5*log(fEigs_e[0]);
1311          double epse1 = 0.5*log(fEigs_e[1]);
1312          double epse2 = 0.5*log(fEigs_e[2]);
1313
1314        double Je = sqrt(fEigs_e.Product());
1315        fEigs_dev = fEigs_e;
1316        fEigs_dev *= pow(Je,-2.0*third);
1317
1318        /*calculate total neq stress*/
1319        fPot[1]->DevStress(fEigs_dev, ftau_NEQ);
1320
1321        ftau_NEQ *=fdmu[i];
1322
1323          *pstress++ =ftau_NEQ[0];
1324          *pstress++ =ftau_NEQ[1];
1325          *pstress++ =ftau_NEQ[2];
1326
1327          ftau += ftau_NEQ;
1328
1329          fInverse.Inverse(fC_Nvn[i]);
1330          fSpectralDecompSpat.PolarDecomp(fF3D, fR, fU, false);
1331
1332          fSpectralDecompSpat.SpectralDecomp_Jacobi(fC_vn[i],false);
1333          fEigs_cvn = fSpectralDecompSpat.Eigenvalues();
1334          fEigs_cvn[0]=sqrt(fEigs_cvn[0]);fEigs_cvn[1]=sqrt(fEigs_cvn[1]);
                fEigs_cvn[2]=sqrt(fEigs_cvn[2]);
1335        fU=fSpectralDecompSpat.EigsToRank2(fEigs_cvn);
1336
1337          dMatrixT U; U.Dimension(3);
1338          fU.ToMatrix(U);
1339          fFv3D.MultAB(fR,U);
1340          fa.MultQBQT(fFv3D, fInverse);
1341
1342          fSpectralDecompSpat.SpectralDecomp_Jacobi(fa, false);
1343          fEigs_a_last = fSpectralDecompSpat.Eigenvalues();
1344          fEigvec1 = fSpectralDecompSpat.Eigenmatrix();
1345
```

```
1346          *pavectr++ = fEigs_a_last[0];
1347          *pavectr++ = fEigs_a_last[1];
1348          *pavectr++ = fEigs_a_last[2];
1349
1350          double epstr0N = 0.5*log(fEigs_a_last[0]);
1351          double epstr1N = 0.5*log(fEigs_a_last[1]);
1352          double epstr2N = 0.5*log(fEigs_a_last[2]);
1353
1354          fInverse.Inverse(fC_Nv[i]);
1355          fSpectralDecompSpat.PolarDecomp(fF3D, fR, fU, false);
1356
1357          fSpectralDecompSpat.SpectralDecomp_Jacobi(fC_v[i],false);
1358          fEigs_cvn = fSpectralDecompSpat.Eigenvalues();
1359          fEigs_cvn[0]=sqrt(fEigs_cvn[0]);fEigs_cvn[1]=sqrt(fEigs_cvn[1]);
                 fEigs_cvn[2]=sqrt(fEigs_cvn[2]);
1360          fU=fSpectralDecompSpat.EigsToRank2(fEigs_cvn);
1361
1362          fU.ToMatrix(U);
1363          fFv3D.MultAB(fR,U);
1364          fa.MultQBQT(fFv3D, fInverse);
1365
1366          fSpectralDecompSpat.SpectralDecomp_Jacobi(fa, false);
1367          fEigs_a = fSpectralDecompSpat.Eigenvalues();
1368          fEigvec1 = fSpectralDecompSpat.Eigenmatrix();
1369          *pavec++ = fEigs_a[0];
1370          *pavec++ = fEigs_a[1];
1371          *pavec++ = fEigs_a[2];
1372
1373          double epse0N = 0.5*log(fEigs_a[0]);
1374          double epse1N = 0.5*log(fEigs_a[1]);
1375          double epse2N = 0.5*log(fEigs_a[2]);
1376
1377        double Ja = sqrt(fEigs_a.Product());
1378        fEigs_a_dev = fEigs_a;
1379        fEigs_a_dev *= pow(Ja,-2.0*third);
1380
1381          fPot[2]->DevStress(fEigs_a_dev, ftau_BACK, fT0);
1382        ftau_BACK *= fHNratio*fdmu[i];
1383
1384          *pbackstress++ =ftau_BACK[0];
1385          *pbackstress++ =ftau_BACK[1];
1386          *pbackstress++ =ftau_BACK[2];
1387
1388          fsback += ftau_BACK;
1389
1390          ftau_FLOW[0] = prefactor*ftau_NEQ[0]-prefactor2*ftau_BACK[0];
1391          ftau_FLOW[1] = prefactor*ftau_NEQ[1]-prefactor2*ftau_BACK[1];
1392          ftau_FLOW[2] = prefactor*ftau_NEQ[2]-prefactor2*ftau_BACK[2];
1393
1394          *pflowstress++ = ftau_FLOW[0];
1395          *pflowstress++ = ftau_FLOW[1];
1396          *pflowstress++ = ftau_FLOW[2];
1397
1398          fzeta += ftau_FLOW;
1399
1400          ftau_N = ftau_BACK;
```

```
1401        ftau_N *= prefactor2;
1402
1403        *pNstress++ = ftau_N[0];
1404        *pNstress++ = ftau_N[1];
1405        *pNstress++ = ftau_N[2];
1406
1407        fNstress += ftau_N;
1408
1409        double plasticwk = -(ftau_NEQ[0]*(epse0-epstr0)+ ftau_NEQ[1]*(
                epse1-epstr1) + ftau_NEQ[2]*(epse2-epstr2));
1410        plasticw += plasticwk;
1411
1412        double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK[1]*(
                epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
1413        backw += backwk;
1414    }
1415
1416     /*update  smag*/
1417     double zeta0n = fzeta[0];
1418     double zeta1n = fzeta[1];
1419     double zeta2n = fzeta[2];
1420
1421     double smag = sqrt(0.5*(zeta0n*zeta0n+zeta1n*zeta1n+zeta2n*zeta2n));
1422
1423
1424     double N0n = fNstress[0];
1425     double N1n = fNstress[1];
1426     double N2n = fNstress[2];
1427
1428     double sbmag = sqrt(0.5*(N0n*N0n+N1n*N1n+N2n*N2n));
1429
1430    /*calc viscosity functions*/
1431    double ietabar = StressRelaxationFunc(Tn, smag);
1432     double itaubar = StructuralRelaxationFunc(Tn, ScTn);
1433
1434     double ietabarN = ReptationRelaxationFuncS(Tn, sbmag);
1435     double itaubarN = ReptationRelaxationFuncT(Tn);
1436
1437     double lm0n=sqrt(fEigs[0]);
1438     double lm1n=sqrt(fEigs[1]);
1439     double lm2n=sqrt(fEigs[2]);
1440     double lmcn=sqrt(lm0n*lm0n+lm1n*lm1n+lm2n*lm2n);
1441
1442     double alpha0n=alphakf(lm0n,lmcn);
1443     double alpha1n=alphakf(lm1n,lmcn);
1444     double alpha2n=alphakf(lm2n,lmcn);
1445
1446     double alphamax=fmax(alpha0n,alpha1n);
1447     alphamax=fmax(alphamax,alpha2n);
1448     double alphan=0.5*pi-alphamax;
1449
1450     double alphafuncn=falpha(alphan);
1451
1452     double dalphadl0n,dalphadl1n,dalphadl2n;
1453
1454     if (alphamax == alpha0n)
```

```
1455    {
1456       dalphad10n=sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
1457         dalphad11n=-lm0n*lm1n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
1458         dalphad12n=-lm0n*lm2n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
1459    }
1460    else if (alphamax == alpha1n)
1461    {
1462       dalphad10n=-lm0n*lm1n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
1463         dalphad11n=sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
1464         dalphad12n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
1465    }
1466    else if (alphamax == alpha2n)
1467    {
1468       dalphad10n=-lm0n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
1469         dalphad11n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
1470         dalphad12n=sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
1471    }
1472
1473     double DetabarDs = 0.0;
1474     if(smag > kSmall)
1475     {
1476         double x = (fQS*smag)/(Tn);
1477         DetabarDs = (x*cosh(x) - sinh(x))/x/smag;
1478     }
1479
1480     double DetabarDsN = 0.0;
1481     if(sbmag > kSmall)
1482     {
1483         double x = (fQSb*sbmag)/(Tn);
1484         DetabarDsN = (x*cosh(x) - sinh(x))/x/sbmag;
1485     }
1486
1487     double DetaSDScTn = fBB/Tn/ScTn/ScTn;
1488
1489    fKAB = 0.0;      /*DRes/Depsilon^e_Bj*/
1490     fKTer = 0.0;     /*DRdel/Depsilon^e_Bj*/
1491     fKrTe = 0.0;     /*DRes/DTe_l*/
1492     fKaAB = 0.0;  /*DfRes/Depsilon^e_Bk*/
1493     fKaTe = 0.0;  /*DRes/DTe_l*/
1494     fKTea = 0.0;/*DRdel/Depsilon^e_Bk*/
1495     fKdel = 0.0;   /*DRdel/DTel*/
1496     fKar=0.0;
1497     fKra=0.0;
1498
1499    fGA0 = 0.0;
1500    fGA1 = 0.0;
1501    fGA2 = 0.0;
1502
1503       fGB0 = 0.0;
1504    fGB1 = 0.0;
1505    fGB2 = 0.0;
1506
1507     fGC0 = 0.0;
1508    fGC1 = 0.0;
1509    fGC2 = 0.0;
1510
```

```
1511    Modulus1 = 0.0;
1512     Modulus2 = 0.0;
1513
1514     ple = fle.Pointer();
1515     pltr = fl_tr.Pointer();
1516     pstress = fstressk.Pointer();
1517     pbackstress = fbackstressk.Pointer();
1518     pflowstress = fflowstressk.Pointer();
1519     pavectr = favec_tr.Pointer();
1520     pavec = favec.Pointer();
1521
1522    for(int k=0; k<fNumS; k++)
1523    {
1524          double epstr0 = 0.5*log(*pltr++);
1525          double epstr1 = 0.5*log(*pltr++);
1526          double epstr2 = 0.5*log(*pltr++);
1527
1528          fEigs_dev[0] = *ple++;
1529          fEigs_dev[1] = *ple++;
1530          fEigs_dev[2] = *ple++;
1531
1532          double epse0 = 0.5*log(fEigs_dev[0]);
1533          double epse1 = 0.5*log(fEigs_dev[1]);
1534          double epse2 = 0.5*log(fEigs_dev[2]);
1535
1536          /*calculate neq stress tauk*/
1537          double Je = sqrt(fEigs_dev[0]*fEigs_dev[1]*fEigs_dev[2]);
1538          fEigs_dev *= pow(Je,-2.0*third);
1539          fPot[1]->DevMod(fEigs_dev,fDtauDe_NEQ);
1540
1541          /*moduli*/
1542          /*viscosity and its derivatives*/
1543          double muk=fdmu[k];
1544          double itauSk = itaubar/ftimesS[k];
1545          double ietaS0k = itauSk/muk;
1546          double ietaSk = ietaS0k*ietabar;
1547
1548          double mukN=fHNratio*fdmu[k];
1549          double itauSkN = itaubarN/(ftimesS[k]*fshift);
1550          double ietaS0kN = itauSkN/mukN;
1551          double ietaSkN = ietaS0kN*ietabarN;
1552
1553          double dk00 = muk*iJ*fDtauDe_NEQ(0,0);
1554          double dk11 = muk*iJ*fDtauDe_NEQ(1,1);
1555          double dk22 = muk*iJ*fDtauDe_NEQ(2,2);
1556          double dk12 = muk*iJ*fDtauDe_NEQ(1,2);
1557          double dk02 = muk*iJ*fDtauDe_NEQ(0,2);
1558          double dk01 = muk*iJ*fDtauDe_NEQ(0,1);
1559
1560          double a0k = *pavec++;
1561          double a1k = *pavec++;
1562          double a2k = *pavec++;
1563
1564          double epse0N = 0.5*log(a0k);
1565          double epse1N = 0.5*log(a1k);
1566          double epse2N = 0.5*log(a2k);
```

```
1567
1568        double epstr0N = 0.5*log(*pavectr++);
1569        double epstr1N = 0.5*log(*pavectr++);
1570        double epstr2N = 0.5*log(*pavectr++);
1571
1572        fEigs_a[0] = a0k;
1573        fEigs_a[1] = a1k;
1574        fEigs_a[2] = a2k;
1575
1576        double Ja = sqrt(fEigs_a.Product());
1577        fEigs_a_dev = fEigs_a;
1578        fEigs_a_dev *= pow(Ja,-2.0*third);
1579        fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
1580       fDtauDe_BACK *= fHNratio*fdmu[k];
1581
1582        double dN00k = prefactor2*fDtauDe_BACK(0,0);
1583        double dN01k = prefactor2*fDtauDe_BACK(0,1);
1584        double dN02k = prefactor2*fDtauDe_BACK(0,2);
1585        double dN10k = prefactor2*fDtauDe_BACK(0,1);
1586        double dN11k = prefactor2*fDtauDe_BACK(1,1);
1587        double dN12k = prefactor2*fDtauDe_BACK(1,2);
1588        double dN20k = prefactor2*fDtauDe_BACK(0,2);
1589        double dN21k = prefactor2*fDtauDe_BACK(1,2);
1590        double dN22k = prefactor2*fDtauDe_BACK(2,2);
1591
1592        double tauk0 =  *pstress++;
1593        double tauk1 =  *pstress++;
1594        double tauk2 =  *pstress++;
1595        double Mk0n = (tauk0)*prefactor;
1596        double Mk1n = (tauk1)*prefactor;
1597        double Mk2n = (tauk2)*prefactor;
1598
1599        double taukback0 =  *pbackstress++;
1600        double taukback1 =  *pbackstress++;
1601        double taukback2 =  *pbackstress++;
1602        double Nk0n = (taukback0)*prefactor2;
1603        double Nk1n = (taukback1)*prefactor2;
1604        double Nk2n = (taukback2)*prefactor2;
1605
1606        double zetak0 =  *pflowstress++;
1607        double zetak1 =  *pflowstress++;
1608        double zetak2 =  *pflowstress++;
1609
1610        /*stiffness matrix*/
1611        int a = k*3;
1612       fKaAB(a,a) = 1.0 +0.5*dt*ietaSkN/alphafuncn*dN00k;
1613       fKaAB(a,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN01k;
1614       fKaAB(a,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN02k;
1615       fKaAB(a+1,a) = 0.5*dt*ietaSkN/alphafuncn*dN10k;
1616       fKaAB(a+1,a+1) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN11k;
1617       fKaAB(a+1,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN12k;
1618       fKaAB(a+2,a) = 0.5*dt*ietaSkN/alphafuncn*dN20k;
1619       fKaAB(a+2,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN21k;
1620       fKaAB(a+2,a+2) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN22k;
1621
1622       fKar(a,a) = 1.0;
```

```
1623        fKar(a,a+1) = 0.0;
1624        fKar(a,a+2) = 0.0;
1625        fKar(a+1,a) = 0.0;
1626        fKar(a+1,a+1) = 1.0;
1627        fKar(a+1,a+2) = 0.0;
1628        fKar(a+2,a) = 0.0;
1629        fKar(a+2,a+1) = 0.0;
1630        fKar(a+2,a+2) = 1.0;
1631
1632        fKra(a,a) = -0.5*dt*ietaSk*dN00k;
1633        fKra(a,a+1) = -0.5*dt*ietaSk*dN01k;
1634        fKra(a,a+2) = -0.5*dt*ietaSk*dN02k;
1635        fKra(a+1,a) = -0.5*dt*ietaSk*dN10k;
1636        fKra(a+1,a+1) = -0.5*dt*ietaSk*dN11k;
1637        fKra(a+1,a+2) = -0.5*dt*ietaSk*dN12k;
1638        fKra(a+2,a) = -0.5*dt*ietaSk*dN20k;
1639        fKra(a+2,a+1) = -0.5*dt*ietaSk*dN21k;
1640        fKra(a+2,a+2) = -0.5*dt*ietaSk*dN22k;
1641
1642        fKAB(a,a) = 1.0 + 0.5*dt*ietaSk*dk00*prefactor/iJ;
1643        fKAB(a,a+1) = 0.5*dt*ietaSk*dk01*prefactor/iJ;
1644        fKAB(a,a+2) = 0.5*dt*ietaSk*dk02*prefactor/iJ;
1645        fKAB(a+1,a) = 0.5*dt*ietaSk*dk01*prefactor/iJ;
1646        fKAB(a+1,a+1) = 1.0 + 0.5*dt*ietaSk*dk11*prefactor/iJ;
1647        fKAB(a+1,a+2) = 0.5*dt*ietaSk*dk12*prefactor/iJ;
1648        fKAB(a+2,a) = 0.5*dt*ietaSk*dk02*prefactor/iJ;
1649        fKAB(a+2,a+1) = 0.5*dt*ietaSk*dk12*prefactor/iJ;
1650        fKAB(a+2,a+2) = 1.0 + 0.5*dt*ietaSk*dk22*prefactor/iJ;
1651
1652        double *ple2 = fle.Pointer();
1653        double *paEigVec = favec.Pointer();
1654        for (int l=0; l<fNumS; l++)       /*evolution eqs. are coupled
               in this case because of s (or smag here)*/
1655        {
1656            /*moduli*/
1657            fEigs_dev[0] = *ple2++;
1658            fEigs_dev[1] = *ple2++;
1659            fEigs_dev[2] = *ple2++;
1660            double Je2 = sqrt(fEigs_dev[0]*fEigs_dev[1]*fEigs_dev[2]);
1661            fEigs_dev *= pow(Je2,-2.0*third);
1662            fPot[1]->DevMod(fEigs_dev,fDtauDe_NEQ);
1663            double dl00 = fdmu[l]*iJ*fDtauDe_NEQ(0,0);
1664            double dl11 = fdmu[l]*iJ*fDtauDe_NEQ(1,1);
1665            double dl22 = fdmu[l]*iJ*fDtauDe_NEQ(2,2);
1666            double dl12 = fdmu[l]*iJ*fDtauDe_NEQ(1,2);
1667            double dl02 = fdmu[l]*iJ*fDtauDe_NEQ(0,2);
1668            double dl01 = fdmu[l]*iJ*fDtauDe_NEQ(0,1);
1669
1670            double cl0 = 0.0;
1671            double cl1 = 0.0;
1672            double cl2 = 0.0;
1673
1674            double a0l = *paEigVec++;
1675            double a1l = *paEigVec++;
1676            double a2l = *paEigVec++;
1677
```

```
1678            fEigs_a[0] = a01;
1679            fEigs_a[1] = a11;
1680            fEigs_a[2] = a21;
1681
1682            double Ja = sqrt(fEigs_a.Product());
1683            fEigs_a_dev = fEigs_a;
1684            fEigs_a_dev *= pow(Ja,-2.0*third);
1685            fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
1686            fDtauDe_BACK *= fHNratio*fdmu[1];
1687
1688            double dN001 = prefactor2*fDtauDe_BACK(0,0);
1689            double dN011 = prefactor2*fDtauDe_BACK(0,1);
1690            double dN021 = prefactor2*fDtauDe_BACK(0,2);
1691            double dN101 = prefactor2*fDtauDe_BACK(0,1);
1692            double dN111 = prefactor2*fDtauDe_BACK(1,1);
1693            double dN121 = prefactor2*fDtauDe_BACK(1,2);
1694            double dN201 = prefactor2*fDtauDe_BACK(0,2);
1695            double dN211 = prefactor2*fDtauDe_BACK(1,2);
1696            double dN221 = prefactor2*fDtauDe_BACK(2,2);
1697
1698            double d10 = 0.0;
1699            double d11 = 0.0;
1700            double d12 = 0.0;
1701
1702            double e10 = 0.0;
1703            double e11 = 0.0;
1704            double e12 = 0.0;
1705
1706            if(smag>kSmall)
1707            {
1708                c10 = 0.5*(fzeta[0]*d100 + fzeta[1]*d101 + fzeta[2]*d102
                        )*prefactor/iJ/smag;
1709                c11 = 0.5*(fzeta[0]*d101 + fzeta[1]*d111 + fzeta[2]*d112
                        )*prefactor/iJ/smag;
1710                c12 = 0.5*(fzeta[0]*d102 + fzeta[1]*d112 + fzeta[2]*d122
                        )*prefactor/iJ/smag;
1711
1712                d10 = -0.5*(fzeta[0]*dN001 + fzeta[1]*dN101 + fzeta[2]*
                        dN201)/smag;
1713                d11 = -0.5*(fzeta[0]*dN011 + fzeta[1]*dN111 + fzeta[2]*
                        dN211)/smag;
1714                d12 = -0.5*(fzeta[0]*dN021 + fzeta[1]*dN121 + fzeta[2]*
                        dN221)/smag;
1715            }
1716
1717            if(sbmag>kSmall)
1718            {
1719                e10 = 0.5*(fNstress[0]*dN001 + fNstress[1]*dN101 +
                        fNstress[2]*dN201)/sbmag;
1720                e11 = 0.5*(fNstress[0]*dN011 + fNstress[1]*dN111 +
                        fNstress[2]*dN211)/sbmag;
1721                e12 = 0.5*(fNstress[0]*dN021 + fNstress[1]*dN121 +
                        fNstress[2]*dN221)/sbmag;
1722            }
1723
1724            int b = 1*3;
```

```
1725                fKaAB(a,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*Nk0n
                        ;
1726                fKaAB(a,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk0n;
1727                fKaAB(a,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e12*
                        Nk0n;
1728                fKaAB(a+1,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*
                        Nk1n;
1729                fKaAB(a+1,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk1n;
1730                fKaAB(a+1,b+2)+= 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e12*
                        Nk1n;
1731                fKaAB(a+2,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*
                        Nk2n;
1732                fKaAB(a+2,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk2n;
1733                fKaAB(a+2,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e12*
                        Nk2n;
1734
1735                fKra(a,b) +=  0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak0;
1736                fKra(a,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak0;
1737                fKra(a,b+2) += 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak0;
1738                fKra(a+1,b) += 0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak1;
1739                fKra(a+1,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak1;
1740                fKra(a+1,b+2)+= 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak1;
1741                fKra(a+2,b) += 0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak2;
1742                fKra(a+2,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak2;
1743                fKra(a+2,b+2) += 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak2;
1744
1745                fKAB(a,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak0;
1746                fKAB(a,b+1) += 0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak0;
1747                fKAB(a,b+2) +=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak0;
1748                fKAB(a+1,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak1;
1749                fKAB(a+1,b+1) +=  0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak1;
1750                fKAB(a+1,b+2)+=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak1;
1751                fKAB(a+2,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak2;
1752                fKAB(a+2,b+1) +=  0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak2;
1753                fKAB(a+2,b+2) +=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak2;
1754            }
1755            fGA0[a] = 1.0 + 0.5*dt*ietaSk*zetak0*DetaSDScTn*l0f;
1756            fGA0[a+1] =     0.5*dt*ietaSk*zetak1*DetaSDScTn*l0f;
1757            fGA0[a+2] =     0.5*dt*ietaSk*zetak2*DetaSDScTn*l0f;
1758
1759            fGA1[a] =       0.5*dt*ietaSk*zetak0*DetaSDScTn*l1f;
1760            fGA1[a+1] = 1.0 + 0.5*dt*ietaSk*zetak1*DetaSDScTn*l1f;
1761            fGA1[a+2] =       0.5*dt*ietaSk*zetak2*DetaSDScTn*l1f;
1762
1763            fGA2[a] =       0.5*dt*ietaSk*zetak0*DetaSDScTn*l2f;
1764            fGA2[a+1] =     0.5*dt*ietaSk*zetak1*DetaSDScTn*l2f;
1765            fGA2[a+2] = 1.0 + 0.5*dt*ietaSk*zetak2*DetaSDScTn*l2f;
1766
1767            fGC0[a] = 1.0 - 0.5*dt*ietaSkN/alphafuncn*Nk0n*(1.0/(alphan-
                    falphac)+fmpower/alphan)*dalphadl0n*lm0n;
1768            fGC0[a+1] =   - 0.5*dt*ietaSkN/alphafuncn*Nk1n*(1.0/(alphan-
                    falphac)+fmpower/alphan)*dalphadl0n*lm0n;
1769            fGC0[a+2] =   - 0.5*dt*ietaSkN/alphafuncn*Nk2n*(1.0/(alphan-
```

```
            falphac)+fmpower/alphan)*dalphadl0n*lm0n;
1770
1771        fGC1[a] =       - 0.5*dt*ietaSkN/alphafuncn*Nk0n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl1n*lm1n;
1772        fGC1[a+1] = 1.0 - 0.5*dt*ietaSkN/alphafuncn*Nk1n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl1n*lm1n;
1773        fGC1[a+2] =     - 0.5*dt*ietaSkN/alphafuncn*Nk2n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl1n*lm1n;
1774
1775        fGC2[a] =       - 0.5*dt*ietaSkN/alphafuncn*Nk0n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl2n*lm2n;
1776        fGC2[a+1] =     - 0.5*dt*ietaSkN/alphafuncn*Nk1n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl2n*lm2n;
1777        fGC2[a+2] = 1.0 - 0.5*dt*ietaSkN/alphafuncn*Nk2n*(1.0/(alphan-
                falphac)+fmpower/alphan)*dalphadl2n*lm2n;
1778
1779        for(int l=0; l<fNumR; l++)
1780        {
1781            fKTea(l,a) = ffrac3[l]*((dN00k*(epse0N-epstr0N)+dN10k*(
                    epse1N-epstr1N)+dN20k*(epse2N-epstr2N))/prefactor2+
                    taukback0);
1782            fKTea(l,a+1) = ffrac3[l]*((dN01k*(epse0N-epstr0N)+dN11k*(
                    epse1N-epstr1N)+dN21k*(epse2N-epstr2N))/prefactor2+
                    taukback1);
1783            fKTea(l,a+2) = ffrac3[l]*((dN02k*(epse0N-epstr0N)+dN12k*(
                    epse1N-epstr1N)+dN22k*(epse2N-epstr2N))/prefactor2+
                    taukback2);
1784
1785            double M0n = prefactor*ftau[0];
1786            double M1n = prefactor*ftau[1];
1787            double M2n = prefactor*ftau[2];
1788
1789            double N0n = prefactor2*fsback[0];
1790            double N1n = prefactor2*fsback[1];
1791            double N2n = prefactor2*fsback[2];
1792
1793            double zm = 0.5*(zeta0n*M0n+zeta1n*M1n+zeta2n*M2n);
1794            double zn = 0.5*(zeta0n*N0n+zeta1n*N1n+zeta2n*N2n);
1795
1796            double phi = fdalpha[l];
1797
1798            double DsDTek = 0.0;
1799            if (smag>kSmall)
1800                DsDTek = -phi*Tn*fafrac*zm/prefactor/smag/fTfk[l]/fTfk[l
                        ]+phi*Tn*fbfrac*zn/prefactor2/smag/fTfk[l]/fTfk[l];
1801
1802            double DsbDTek = 0.0;
1803            if (sbmag>kSmall)
1804                DsbDTek = -phi*Tn*fbfrac*sbmag/prefactor2/fTfk[l]/fTfk[l
                        ];
1805
1806            fKaTe(a,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk0n*DetabarDsN*
                    DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*phi*Tn/fTfk[l]/
                    fTfk[l]*taukback0;
1807            fKaTe(a+1,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk1n*DetabarDsN*
                    DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*phi*Tn/fTfk[l]/
```

```
                fTfk[l]*taukback1;
1808            fKaTe(a+2,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk2n*DetabarDsN*
                    DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*phi*Tn/fTfk[l]/
                    fTfk[l]*taukback2;
1809
1810            fKTer(l,a) = ffrac2[l]*(dk00/iJ*(epse0-epstr0) + tauk0 +
                    dk01/iJ*(epse1-epstr1)+dk02/iJ*(epse2-epstr2))+ffrac3[l
                    ]*taukback0;
1811            fKTer(l,a+1) = ffrac2[l]*(dk01/iJ*(epse0-epstr0) + tauk1 +
                    dk11/iJ*(epse1-epstr1)+dk12/iJ*(epse2-epstr2))+ffrac3[l
                    ]*taukback1;
1812            fKTer(l,a+2) = ffrac2[l]*(dk02/iJ*(epse0-epstr0) + tauk2 +
                    dk12/iJ*(epse1-epstr1)+dk22/iJ*(epse2-epstr2))+ffrac3[l
                    ]*taukback2;
1813
1814            fKrTe(a,l) =    0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*fdeltac0
                    /fTfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/ietabar)*zetak0
                    +0.5*dt*ietaSk*(-Mk0n*phi*Tn*fafrac/prefactor/fTfk[l]/
                    fTfk[l]+Nk0n*phi*Tn*fbfrac/prefactor2/fTfk[l]/fTfk[l]);
1815            fKrTe(a+1,l) = 0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*fdeltac0
                    /fTfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/ietabar)*zetak1
                    +0.5*dt*ietaSk*(-Mk1n*phi*Tn*fafrac/prefactor/fTfk[l]/
                    fTfk[l]+Nk1n*phi*Tn*fbfrac/prefactor2/fTfk[l]/fTfk[l]);
1816            fKrTe(a+2,l) = 0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*fdeltac0
                    /fTfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/ietabar)*zetak2
                    +0.5*dt*ietaSk*(-Mk2n*phi*Tn*fafrac/prefactor/fTfk[l]/
                    fTfk[l]+Nk2n*phi*Tn*fbfrac/prefactor2/fTfk[l]/fTfk[l]);
1817
1818        }
1819    }
1820
1821    for(int k=0; k<fNumR; k++)
1822    {
1823        double itauRk = itaubar/ftimesR[k];
1824        fKdel(k,k) = 1.0 + dt*itauRk-ffrac1[k]*flatenth[k]/fTfk[k]+
                fafrac*ffrac1[k]*ffrac1[k]*fdeltac1*plasticw+ffrac1[k]*
                ffrac1[k]*fdeltac1*flatenth[k]-fbfrac*ffrac1[k]*ffrac1[k]*
                fdeltac1*backw;
1825        for (int l=0; l<fNumR; l++)
1826        {
1827            double phi=fdalpha[l];
1828            double DtaubarDTfk = -fBB/Tn/ScTn/ScTn*(phi*fdeltac0/fTfk[l
                    ]+fdeltac1*phi);
1829            fKdel(k,l) += -dt*itauRk*(fTfk[k]-Tn)*DtaubarDTfk;
                        }
1830
1831        fGB0[k] = dt*itauRk*(fTfk[k]-Tn)*DetaSDScTn*l0f+ffrac1[k]*fTfk[k
                ]/fT0*(fDtauDe_EQ(0,0)*depsilon0+fDtauDe_EQ(1,0)*depsilon1+
                fDtauDe_EQ(2,0)*depsilon2 + ftau_EQ[0])+ffrac2[k]*ftau[0]+
                ffrac3[k]*fsback[0];
1832
1833        fGB1[k] = dt*itauRk*(fTfk[k]-Tn)*DetaSDScTn*l1f+ffrac1[k]*fTfk[k
                ]/fT0*(fDtauDe_EQ(0,1)*depsilon0+fDtauDe_EQ(1,1)*depsilon1+
                fDtauDe_EQ(2,1)*depsilon2 + ftau_EQ[1])+ffrac2[k]*ftau[1]+
                ffrac3[k]*fsback[1];
1834
```

```
1835            fGB2[k] = dt*itauRk*(fTfk[k]-Tn)*DetaSDScTn*12f+ffrac1[k]*fTfk[k
                    ]/fT0*(fDtauDe_EQ(0,2)*depsilon0+fDtauDe_EQ(1,2)*depsilon1+
                    fDtauDe_EQ(2,2)*depsilon2 + ftau_EQ[2])+ffrac2[k]*ftau[2]+
                    ffrac3[k]*fsback[2];
1836        }
1837
1838        fK1=fKdel;
1839        fK2=fKAB;
1840        fKdel.Inverse();
1841        fKAB.Inverse();
1842        fKaAB.Inverse();
1843
1844        fKdel2.MultABC(fKTea, fKaAB, fKaTe);
1845        fKAB2.MultABC(fKra, fKaAB, fKar);
1846        fKrTe2.MultABC(fKra, fKaAB, fKaTe);
1847        fKTer2.MultABC(fKTea, fKaAB, fKar);
1848
1849        fK1.AddScaled(-1.0, fKdel2);
1850        fK2.AddScaled(-1.0, fKAB2);
1851        fKrTe.AddScaled(-1.0, fKrTe2);
1852        fKTer.AddScaled(-1.0, fKTer2);
1853
1854        fInverse2.MultAB(fKTea,fKaAB);
1855        fInverse2.Multx(fGC0,fG10);
1856        fInverse2.Multx(fGC1,fG11);
1857        fInverse2.Multx(fGC2,fG12);
1858        fG10 *= -1.0;
1859        fG11 *= -1.0;
1860        fG12 *= -1.0;
1861        fG10 += fGB0;
1862        fG11 += fGB1;
1863        fG12 += fGB2;
1864
1865        fInverse3.MultAB(fKra,fKaAB);
1866        fInverse3.Multx(fGC0,fG20);
1867        fInverse3.Multx(fGC1,fG21);
1868        fInverse3.Multx(fGC2,fG22);
1869        fG20 *= -1.0;
1870        fG21 *= -1.0;
1871        fG22 *= -1.0;
1872        fG20 += fGA0;
1873        fG21 += fGA1;
1874        fG22 += fGA2;
1875
1876        fK3 = fK1;
1877        fK4 = fK2;
1878
1879        fK1.Inverse();
1880        fK2.Inverse();
1881        fK3temp.MultABC(fKTer, fK2, fKrTe);
1882        fK4temp.MultABC(fKrTe, fK1, fKTer);
1883
1884        fK3.AddScaled(-1.0, fK3temp);
1885        fK4.AddScaled(-1.0, fK4temp);
1886
1887        fInverse4.MultAB(fKTer,fK2);
```

```
1888      fInverse4.Multx(fG20,fG10temp);
1889      fInverse4.Multx(fG21,fG11temp);
1890      fInverse4.Multx(fG22,fG12temp);
1891      fG10temp *= -1.0;
1892      fG11temp *= -1.0;
1893      fG12temp *= -1.0;
1894      fG10temp += fG10;
1895      fG11temp += fG11;
1896      fG12temp += fG12;
1897
1898      fInverse5.MultAB(fKrTe,fK1);
1899      fInverse5.Multx(fG10,fG20temp);
1900      fInverse5.Multx(fG11,fG21temp);
1901      fInverse5.Multx(fG12,fG22temp);
1902      fG20temp *= -1.0;
1903      fG21temp *= -1.0;
1904      fG22temp *= -1.0;
1905      fG20temp += fG20;
1906      fG21temp += fG21;
1907      fG22temp += fG22;
1908
1909      fK1=fK3;
1910      fK2=fK4;
1911
1912     fK3.LinearSolve(fG10temp);
1913     fK3 = fK1;
1914     fK3.LinearSolve(fG11temp);
1915     fK3 = fK1;
1916     fK3.LinearSolve(fG12temp);
1917     fK3 = fK1;
1918
1919      fK4.LinearSolve(fG20temp);
1920     fK4 = fK2;
1921      fK4.LinearSolve(fG21temp);
1922     fK4 = fK2;
1923      fK4.LinearSolve(fG22temp);
1924     fK4 = fK2;
1925
1926      for (int k =0; k<fNumR; k++)
1927     {
1928        Modulus1(k,0) = fG10temp[k];
1929        Modulus1(k,1) = fG11temp[k];
1930        Modulus1(k,2) = fG12temp[k];
1931     }
1932     for (int k =0; k<3*fNumS; k++)
1933     {
1934        Modulus2(k,0) = fG20temp[k];
1935        Modulus2(k,1) = fG21temp[k];
1936        Modulus2(k,2) = fG22temp[k];
1937     }
1938
1939 }
1940
1941 void ET_multi::Compute_a(const dArrayT& eigs_n, const dArrayT& eigs,
      const dArrayT& eigs_tr, const dArrayT& DScDe, const dArrayT&
      stretche,const dArrayT& stressk, dArrayT& backstressk, const dArrayT
```

```
    & Tfk_n, dArrayT& Tfk, dArrayT& avec, const dArrayT& avec_tr)
{
     double ctol = 1.00e-8;
    int maxiter = 300;

    /*time step*/
    const double dt = fFSMatSupport->TimeStep();

    /*temperature and temperature step*/
    const double Tn = Compute_Temperature();
     const double Tp = Compute_Temperature_last();

     double l0_last = eigs_n[0];
     double l1_last = eigs_n[1];
     double l2_last = eigs_n[2];
     double Jp = sqrt(l0_last*l1_last*l2_last);

     double l0 = eigs[0];
     double l1 = eigs[1];
     double l2 = eigs[2];
     double J = sqrt(l0*l1*l2);
     double iJ = 1.0/J;

     double depsilon0=(0.5*log(l0)-0.5*log(l0_last));
     double depsilon1=(0.5*log(l1)-0.5*log(l1_last));
     double depsilon2=(0.5*log(l2)-0.5*log(l2_last));

     const double* ple = stretche.Pointer();
     const double* pltr = eigs_tr.Pointer();
     const double* pstress = stressk.Pointer();
     double* pbackstress = backstressk.Pointer();
     double* pNstress = fNstressk.Pointer();
     double* pflowstress = fflowstressk.Pointer();
     const double* pavectr = avec_tr.Pointer();
     double* pavec = avec.Pointer();

     ftau = 0.0;
     fsback = 0.0;
    for (int i = 0; i < fNumS; i++)
    {
       fEigs_e[0] = *ple++;
       fEigs_e[1] = *ple++;
       fEigs_e[2] = *ple++;

       double Je = sqrt(fEigs_e.Product());
       fEigs_dev = fEigs_e;
       fEigs_dev *= pow(Je,-2.0*third);

         ftau_NEQ[0] =*pstress++;
         ftau_NEQ[1] =*pstress++;
         ftau_NEQ[2] =*pstress++;

         ftau+= ftau_NEQ;

         fEigs_a_last[0] = *pavectr++;
      fEigs_a_last[1] = *pavectr++;
```

```
1997        fEigs_a_last[2] = *pavectr++;
1998
1999          fEigs_a[0] = *pavec++;
2000        fEigs_a[1] = *pavec++;
2001        fEigs_a[2] = *pavec++;
2002
2003        double Ja = sqrt(fEigs_a.Product());
2004        fEigs_a_dev = fEigs_a;
2005        fEigs_a_dev *= pow(Ja,-2.0*third);
2006
2007          ftau_BACK[0] = *pbackstress++;
2008          ftau_BACK[1] = *pbackstress++;
2009          ftau_BACK[2] = *pbackstress++;
2010
2011          fsback += ftau_BACK;
2012
2013      }
2014
2015       /*compute effective temperatures*/
2016       Compute_Tei(eigs_n, eigs, eigs_tr, DScDe, stretche, stressk,
              backstressk, Tfk_n, Tfk, avec, avec_tr);
2017
2018       double ScTn = 0.0;
2019       double prefactor = 1.0;
2020       double prefactor2 = 1.0;
2021       double Tfn = 0.0;
2022       double Tfp = 0.0;
2023       double plasticw = 0.0; //initially zero because epe-eptr = 0.0;
2024       double backw = 0.0;
2025
2026      for(int k=0; k<fNumR; k++)
2027      {
2028         double phi=fdalpha[k];
2029           fdeltac[k]=capacityc(Tfk[k]);
2030           ffrac1[k]=1.0/fdeltac[k];
2031           ffrac2[k]=fafrac/fdeltac[k];
2032           ffrac3[k]=fbfrac/fdeltac[k];
2033           Tfn += (Tfk[k]*phi);
2034           Tfp += (Tfk_n[k]*phi);
2035           ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-fT2);
2036           prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
2037           prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
2038           flatenth[k] =  Tfk[k]/fT0*(DScDe[0]*depsilon0+DScDe[1]*depsilon1
                 +DScDe[2]*depsilon2);
2039       }
2040
2041      fEigs_dev = eigs;
2042      fEigs_dev *= pow(J, -2.0*third);
2043       double energy = fPot[0]->Energy(fEigs_dev, J);
2044       energy -= fPot[0]->MeanEnergy(J);
2045       ScTn -= energy/fT0;
2046
2047       fNstress=0.0;
2048       pstress = stressk.Pointer();
2049       pbackstress = backstressk.Pointer();
2050       pNstress = fNstressk.Pointer();
```

```
2051     ple = stretche.Pointer();
2052     pltr = eigs_tr.Pointer();
2053     pavectr = avec_tr.Pointer();
2054     pavec = avec.Pointer();
2055 
2056     for (int i = 0; i < fNumS; i++)
2057    {
2058         double epse0 = 0.5*log(*ple++);
2059         double epse1 = 0.5*log(*ple++);
2060         double epse2 = 0.5*log(*ple++);
2061 
2062         double epstr0 = 0.5*log(*pltr++);
2063         double epstr1 = 0.5*log(*pltr++);
2064         double epstr2 = 0.5*log(*pltr++);
2065 
2066         double epse0N = 0.5*log(*pavec++);
2067         double epse1N = 0.5*log(*pavec++);
2068         double epse2N = 0.5*log(*pavec++);
2069 
2070         double epstr0N = 0.5*log(*pavectr++);
2071         double epstr1N = 0.5*log(*pavectr++);
2072         double epstr2N = 0.5*log(*pavectr++);
2073 
2074         ftau_NEQ[0] = *pstress++;
2075         ftau_NEQ[1] = *pstress++;
2076         ftau_NEQ[2] = *pstress++;
2077 
2078         ftau_BACK[0] = *pbackstress++;
2079         ftau_BACK[1] = *pbackstress++;
2080         ftau_BACK[2] = *pbackstress++;
2081 
2082         ftau_N = ftau_BACK;
2083         ftau_N *= prefactor2;
2084 
2085         *pNstress++ = ftau_N[0];
2086         *pNstress++ = ftau_N[1];
2087         *pNstress++ = ftau_N[2];
2088 
2089         fNstress += ftau_N;
2090 
2091         double plasticwk = -(ftau_NEQ[0]*(epse0-epstr0)+ ftau_NEQ[1]*(
                epse1-epstr1) + ftau_NEQ[2]*(epse2-epstr2));
2092         plasticw += plasticwk;
2093 
2094         double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK[1]*(
                epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
2095         backw += backwk;
2096     }
2097 
2098     double N0n = fNstress[0];
2099     double N1n = fNstress[1];
2100     double N2n = fNstress[2];
2101 
2102     double sbmag = sqrt(0.5*(N0n*N0n+N1n*N1n+N2n*N2n));
2103 
2104    /*calc viscosity functions*/
```

```
2105    double ietabarN = ReptationRelaxationFuncS(Tn, sbmag);
2106     double itaubarN = ReptationRelaxationFuncT(Tn);
2107     double itaubar = StructuralRelaxationFunc(Tn, ScTn);
2108    double DetabarDsN = 0.0;
2109     if(sbmag > kSmall)
2110     {
2111         double x = (fQSb*sbmag)/(Tn);
2112         DetabarDsN = (x*cosh(x) - sinh(x))/x/sbmag;
2113     }
2114
2115     double lm0n=sqrt(eigs[0]);
2116     double lm1n=sqrt(eigs[1]);
2117     double lm2n=sqrt(eigs[2]);
2118     double lmcn=sqrt(lm0n*lm0n+lm1n*lm1n+lm2n*lm2n);
2119
2120     double alpha0n=alphakf(lm0n,lmcn);
2121     double alpha1n=alphakf(lm1n,lmcn);
2122     double alpha2n=alphakf(lm2n,lmcn);
2123
2124     double alphamax=fmax(alpha0n,alpha1n);
2125     alphamax=fmax(alphamax,alpha2n);
2126     double alphan=0.5*pi-alphamax;
2127
2128     double alphafuncn=falpha(alphan);
2129
2130     double dalphadl0n,dalphadl1n,dalphadl2n;
2131
2132     if (alphamax == alpha0n)
2133    {
2134        dalphadl0n=sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2135          dalphadl1n=-lm0n*lm1n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2136          dalphadl2n=-lm0n*lm2n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2137    }
2138    else if (alphamax == alpha1n)
2139    {
2140        dalphadl0n=-lm0n*lm1n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2141          dalphadl1n=sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2142          dalphadl2n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2143    }
2144    else if (alphamax == alpha2n)
2145    {
2146        dalphadl0n=-lm0n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2147          dalphadl1n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2148          dalphadl2n=sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2149    }
2150
2151    /*calc residual*/
2152    /*re-assign pointers*/
2153     pstress = stressk.Pointer();
2154     pbackstress = backstressk.Pointer();
2155     pNstress = fNstressk.Pointer();
2156     ple = stretche.Pointer();
2157     pltr = eigs_tr.Pointer();
2158     pavectr = avec_tr.Pointer();
2159     pavec = avec.Pointer();
2160
```

```
2161    double* pra = fResa.Pointer();
2162    fResa = 0.0;
2163    double ra0, ra1, ra2, tol;
2164    double temp = 0.0;
2165
2166    for (int k = 0; k < fNumS; k++)
2167    {
2168         double epse0N = 0.5*log(*pavec++);
2169         double epse1N = 0.5*log(*pavec++);
2170         double epse2N = 0.5*log(*pavec++);
2171
2172         double epstr0N = 0.5*log(*pavectr++);
2173         double epstr1N = 0.5*log(*pavectr++);
2174         double epstr2N = 0.5*log(*pavectr++);
2175
2176         ftau_N[0] = *pNstress++;
2177         ftau_N[1] = *pNstress++;
2178         ftau_N[2] = *pNstress++;
2179
2180         double s0 = ftau_N[0];
2181         double s1 = ftau_N[1];
2182         double s2 = ftau_N[2];
2183
2184       /*viscosity*/
2185         double mukN=fHNratio*fdmu[k];
2186         double itauSkN = itaubarN/(ftimesS[k]*fshift);
2187         double ietaS0kN = itauSkN/mukN;
2188         double ietaSkN = ietaS0kN*ietabarN;
2189       /*calc residual*/
2190         ra0 = epse0N - epstr0N + 0.5*dt*ietaSkN*s0/alphafuncn;
2191         ra1 = epse1N - epstr1N + 0.5*dt*ietaSkN*s1/alphafuncn;
2192         ra2 = epse2N - epstr2N + 0.5*dt*ietaSkN*s2/alphafuncn;
2193
2194       temp += ra0*ra0 + ra1*ra1+ ra2*ra2;
2195       *pra++ = ra0;
2196       *pra++ = ra1;
2197       *pra++ = ra2;
2198     }
2199    tol = sqrt(temp);
2200    double tol0 = tol;
2201    double reltol =tol0;
2202    int iter = 0;
2203
2204     while (tol > ctol  && iter < maxiter)
2205    {
2206       iter++;
2207         fKaAB2 = 0.0;
2208       fKaAB = 0.0; /*DfRes/Depsilon^e_Bk*/
2209         fKaTe = 0.0; /*DRes/DTe_1*/
2210         fKTea = 0.0;/*DRdel/Depsilon^e_Bk*/
2211       fKdel = 0.0;  /*DRdel/DTel*/
2212         pstress = stressk.Pointer();
2213         pbackstress = backstressk.Pointer();
2214         pNstress = fNstressk.Pointer();
2215         ple = stretche.Pointer();
2216         pltr = eigs_tr.Pointer();
```

```
2217          pavectr = avec_tr.Pointer();
2218          pavec = avec.Pointer();
2219        double DetabarDsN = 0.0;
2220          if(sbmag > kSmall)
2221          {
2222              double x = (fQSb*sbmag)/(Tn);
2223              DetabarDsN = (x*cosh(x) - sinh(x))/x/sbmag;
2224          }
2225
2226        for(int k=0; k<fNumS; k++)
2227        {
2228              double a0k = *pavec++;
2229              double a1k = *pavec++;
2230              double a2k = *pavec++;
2231
2232              double epse0N = 0.5*log(a0k);
2233              double epse1N = 0.5*log(a1k);
2234              double epse2N = 0.5*log(a2k);
2235
2236              double epstr0N = 0.5*log(*pavectr++);
2237              double epstr1N = 0.5*log(*pavectr++);
2238              double epstr2N = 0.5*log(*pavectr++);
2239
2240              double mukN=fHNratio*fdmu[k];
2241              double itauSkN = itaubarN/(ftimesS[k]*fshift);
2242              double ietaS0kN = itauSkN/mukN;
2243              double ietaSkN = ietaS0kN*ietabarN;
2244
2245              fEigs_a[0] = a0k;
2246           fEigs_a[1] = a1k;
2247           fEigs_a[2] = a2k;
2248
2249              double Ja = sqrt(fEigs_a.Product());
2250              fEigs_a_dev = fEigs_a;
2251              fEigs_a_dev *= pow(Ja,-2.0*third);
2252              fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
2253           fDtauDe_BACK *= fHNratio*fdmu[k];
2254
2255              double dN00k = prefactor2*fDtauDe_BACK(0,0);
2256              double dN01k = prefactor2*fDtauDe_BACK(0,1);
2257              double dN02k = prefactor2*fDtauDe_BACK(0,2);
2258              double dN10k = prefactor2*fDtauDe_BACK(0,1);
2259              double dN11k = prefactor2*fDtauDe_BACK(1,1);
2260              double dN12k = prefactor2*fDtauDe_BACK(1,2);
2261              double dN20k = prefactor2*fDtauDe_BACK(0,2);
2262              double dN21k = prefactor2*fDtauDe_BACK(1,2);
2263              double dN22k = prefactor2*fDtauDe_BACK(2,2);
2264
2265              double taukback0 =  *pbackstress++;
2266              double taukback1 =  *pbackstress++;
2267              double taukback2 =  *pbackstress++;
2268              double Nk0n = *pNstress++;
2269              double Nk1n = *pNstress++;
2270              double Nk2n = *pNstress++;
2271
2272           /*stiffness matrix*/
```

```
2273            int a = k*3;
2274            fKaAB(a,a) = 1.0 +0.5*dt*ietaSkN/alphafuncn*dN00k;
2275            fKaAB(a,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN01k;
2276            fKaAB(a,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN02k;
2277            fKaAB(a+1,a) = 0.5*dt*ietaSkN/alphafuncn*dN10k;
2278            fKaAB(a+1,a+1) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN11k;
2279            fKaAB(a+1,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN12k;
2280            fKaAB(a+2,a) = 0.5*dt*ietaSkN/alphafuncn*dN20k;
2281            fKaAB(a+2,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN21k;
2282            fKaAB(a+2,a+2) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN22k;
2283
2284               double *paEigVec = avec.Pointer();
2285            for (int l=0; l<fNumS; l++)       /*evolution eqs. are coupled
                   in this case because of s (or smag here)*/
2286            {
2287               /*moduli*/
2288               double a0l = *paEigVec++;
2289                   double a1l = *paEigVec++;
2290                   double a2l = *paEigVec++;
2291
2292                   fEigs_a[0] = a0l;
2293                   fEigs_a[1] = a1l;
2294                   fEigs_a[2] = a2l;
2295
2296                   double Ja = sqrt(fEigs_a.Product());
2297                   fEigs_a_dev = fEigs_a;
2298                   fEigs_a_dev *= pow(Ja,-2.0*third);
2299                   fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
2300                   fDtauDe_BACK *= fHNratio*fdmu[l];
2301
2302                   double dN00l = prefactor2*fDtauDe_BACK(0,0);
2303                   double dN01l = prefactor2*fDtauDe_BACK(0,1);
2304                   double dN02l = prefactor2*fDtauDe_BACK(0,2);
2305                   double dN10l = prefactor2*fDtauDe_BACK(0,1);
2306                   double dN11l = prefactor2*fDtauDe_BACK(1,1);
2307                   double dN12l = prefactor2*fDtauDe_BACK(1,2);
2308                   double dN20l = prefactor2*fDtauDe_BACK(0,2);
2309                   double dN21l = prefactor2*fDtauDe_BACK(1,2);
2310                   double dN22l = prefactor2*fDtauDe_BACK(2,2);
2311
2312                   double d10 = 0.0;
2313                   double d11 = 0.0;
2314                   double d12 = 0.0;
2315
2316                   if(sbmag>kSmall)
2317                   {
2318                       d10 = (fNstress[0]*dN00l + fNstress[1]*dN10l +
                               fNstress[2]*dN20l)/sbmag;
2319                       d11 = (fNstress[0]*dN01l + fNstress[1]*dN11l +
                               fNstress[2]*dN21l)/sbmag;
2320                       d12 = (fNstress[0]*dN02l + fNstress[1]*dN12l +
                               fNstress[2]*dN22l)/sbmag;
2321                   }
2322
2323               int b = l*3;
2324               fKaAB(a,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*d10*
```

```
                Nk0n;
2325            fKaAB(a,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d11*Nk0n;
2326            fKaAB(a,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d12*Nk0n;
2327            fKaAB(a+1,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d10*Nk1n;
2328            fKaAB(a+1,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d11*Nk1n;
2329            fKaAB(a+1,b+2)+= 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d12*Nk1n;
2330            fKaAB(a+2,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d10*Nk2n;
2331            fKaAB(a+2,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*0.5*
                    d11*Nk2n;
2332                fKaAB(a+2,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN
                        *0.5*d12*Nk2n;
2333            }
2334            for(int l=0; l<fNumR; l++)
2335            {
2336                fKTea(l,a) =    ffrac3[l]*((dN00k*(epse0N-epstr0N)+dN10k
                        *(epse1N-epstr1N)+dN20k*(epse2N-epstr2N))/prefactor2
                        +taukback0);
2337                fKTea(l,a+1) = ffrac3[l]*((dN01k*(epse0N-epstr0N)+dN11k
                        *(epse1N-epstr1N)+dN21k*(epse2N-epstr2N))/prefactor2
                        +taukback1);
2338                fKTea(l,a+2) = ffrac3[l]*((dN02k*(epse0N-epstr0N)+dN12k
                        *(epse1N-epstr1N)+dN22k*(epse2N-epstr2N))/prefactor2
                        +taukback2);
2339
2340                double phi = fdalpha[l];
2341
2342                double DsbDTek = 0.0;
2343                if (sbmag>kSmall)
2344                    DsbDTek = -phi*Tn*fbfrac*sbmag/prefactor2/Tfk[l]/Tfk
                            [l];
2345
2346                fKaTe(a,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk0n*DetabarDsN*
                        DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*phi*Tn/Tfk[
                        l]/Tfk[l]*taukback0;
2347                fKaTe(a+1,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk1n*
                        DetabarDsN*DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*
                        phi*Tn/Tfk[l]/Tfk[l]*taukback1;
2348                fKaTe(a+2,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk2n*
                        DetabarDsN*DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*
                        phi*Tn/Tfk[l]/Tfk[l]*taukback2;
2349            }
2350        }
2351
2352        for(int k=0; k<fNumR; k++)
2353      {
2354         /*stiffness matrix*/
2355         double itauRk = itaubar/ftimesR[k];
2356            fKdel(k,k) = 1.0 + dt*itauRk-ffrac1[k]*flatenth[k]/Tfk[k]+
                    fafrac*ffrac1[k]*ffrac1[k]*fdeltac1*plasticw+ffrac1[k]*
                    ffrac1[k]*fdeltac1*flatenth[k]-fbfrac*ffrac1[k]*ffrac1[k
```

```
                ]*fdeltac1*backw;
2357          for (int l=0; l<fNumR; l++)
2358             {
2359                 double phi=fdalpha[l];
2360                 double DtaubarDTfk = -fBB/Tn/ScTn/ScTn*(phi*fdeltac0/Tfk
                        [l]+fdeltac1*phi);
2361             fKdel(k,l) += -dt*itauRk*(Tfk[k]-Tn)*DtaubarDTfk;
2362             }
2363       }
2364
2365       fKdel.Inverse();
2366         fKaAB2.MultABC(fKaTe, fKdel, fKTea);
2367         fKaAB.AddScaled(-1.0, fKaAB2);
2368
2369         /*Solve  and update*/
2370       fKaAB.LinearSolve(fResa);
2371
2372       pavec = avec.Pointer();
2373       pra = fResa.Pointer();
2374       for (int k = 0; k < 3*fNumS; k++)
2375       {
2376          double depeN = -(*pra++);
2377          *pavec++ *= exp(depeN)*exp(depeN);
2378       }
2379         /*update backstresses*/
2380         pavec = avec.Pointer();
2381         pbackstress = backstressk.Pointer();
2382         for (int k = 0; k < fNumS; k++)
2383         {
2384             fEigs_a[0] = *pavec++;
2385             fEigs_a[1] = *pavec++;
2386             fEigs_a[2] = *pavec++;
2387
2388             double Ja = sqrt(fEigs_a.Product());
2389             fEigs_a_dev = fEigs_a;
2390             fEigs_a_dev *= pow(Ja,-2.0*third);
2391
2392             fPot[2]->DevStress(fEigs_a_dev, ftau_BACK, fT0);
2393             ftau_BACK *= fHNratio*fdmu[k];
2394
2395             *pbackstress++ =ftau_BACK[0];
2396             *pbackstress++ =ftau_BACK[1];
2397             *pbackstress++ =ftau_BACK[2];
2398         }
2399
2400       /*update residual*/
2401         /*update Teff related*/
2402         Compute_Tei(eigs_n, eigs, eigs_tr, DScDe, stretche, stressk,
              backstressk, Tfk_n, Tfk, avec, avec_tr);
2403
2404         double ScTn = 0.0;
2405         double prefactor = 1.0;
2406         double prefactor2 = 1.0;
2407         double Tfn = 0.0;
2408
2409         for(int k=0; k<fNumR; k++)
```

```
2410        {
2411            double phi=fdalpha[k];
2412            fdeltac[k]=capacityc(Tfk[k]);
2413            ffrac1[k]=1.0/fdeltac[k];
2414            ffrac2[k]=fafrac/fdeltac[k];
2415            ffrac3[k]=fbfrac/fdeltac[k];
2416            Tfn += (Tfk[k]*phi);
2417            ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-
                    fT2);
2418            prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
2419            prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
2420            flatenth[k] =  Tfk[k]/fT0*(DScDe[0]*depsilon0+DScDe[1]*
                    depsilon1+DScDe[2]*depsilon2);
2421        }
2422
2423        fEigs_dev = eigs;
2424        fEigs_dev *= pow(J, -2.0*third);
2425        energy = fPot[0]->Energy(fEigs_dev, J);
2426        energy -= fPot[0]->MeanEnergy(J);
2427        ScTn -= energy/fT0;
2428
2429        fNstress=0.0;
2430        backw=0.0;
2431        pbackstress = backstressk.Pointer();
2432        pNstress = fNstressk.Pointer();
2433        pavectr = avec_tr.Pointer();
2434        pavec = avec.Pointer();
2435
2436        for (int i = 0; i < fNumS; i++)
2437        {
2438            double epse0N = 0.5*log(*pavec++);
2439            double epse1N = 0.5*log(*pavec++);
2440            double epse2N = 0.5*log(*pavec++);
2441
2442            double epstr0N = 0.5*log(*pavectr++);
2443            double epstr1N = 0.5*log(*pavectr++);
2444            double epstr2N = 0.5*log(*pavectr++);
2445
2446            ftau_BACK[0] = *pbackstress++;
2447            ftau_BACK[1] = *pbackstress++;
2448            ftau_BACK[2] = *pbackstress++;
2449
2450            ftau_N = ftau_BACK;
2451            ftau_N *= prefactor2;
2452
2453            *pNstress++ = ftau_N[0];
2454            *pNstress++ = ftau_N[1];
2455            *pNstress++ = ftau_N[2];
2456
2457            fNstress += ftau_N;
2458
2459            double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK
                    [1]*(epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
2460            backw += backwk;
2461        }
2462
```

```
2463        N0n = fNstress[0];
2464        N1n = fNstress[1];
2465        N2n = fNstress[2];
2466
2467        sbmag = sqrt(0.5*(N0n*N0n+N1n*N1n+N2n*N2n));
2468
2469      /*update viscosity*/
2470        ietabarN = ReptationRelaxationFuncS(Tn, sbmag);
2471        itaubarN = ReptationRelaxationFuncT(Tn);
2472
2473        pbackstress = backstressk.Pointer();
2474        pNstress = fNstressk.Pointer();
2475        pavectr = avec_tr.Pointer();
2476        pavec = avec.Pointer();
2477
2478        pra = fResa.Pointer();
2479        temp = 0.0;
2480
2481        for (int k = 0; k < fNumS; k++)
2482        {
2483            double epse0N = 0.5*log(*pavec++);
2484            double epse1N = 0.5*log(*pavec++);
2485            double epse2N = 0.5*log(*pavec++);
2486
2487            double epstr0N = 0.5*log(*pavectr++);
2488            double epstr1N = 0.5*log(*pavectr++);
2489            double epstr2N = 0.5*log(*pavectr++);
2490
2491            ftau_N[0] = *pNstress++;
2492            ftau_N[1] = *pNstress++;
2493            ftau_N[2] = *pNstress++;
2494
2495            double s0 = ftau_N[0];
2496            double s1 = ftau_N[1];
2497            double s2 = ftau_N[2];
2498
2499            /*viscosity*/
2500            double mukN=fHNratio*fdmu[k];
2501            double itauSkN = itaubarN/(ftimesS[k]*fshift);
2502            double ietaS0kN = itauSkN/mukN;
2503            double ietaSkN = ietaS0kN*ietabarN;
2504            /*calc residual*/
2505            ra0 = epse0N - epstr0N + 0.5*dt*ietaSkN*s0/alphafuncn;
2506            ra1 = epse1N - epstr1N + 0.5*dt*ietaSkN*s1/alphafuncn;
2507            ra2 = epse2N - epstr2N + 0.5*dt*ietaSkN*s2/alphafuncn;
2508
2509            temp += ra0*ra0 + ra1*ra1+ ra2*ra2;
2510            *pra++ = ra0;
2511            *pra++ = ra1;
2512            *pra++ = ra2;
2513        }
2514
2515        tol = sqrt(temp);
2516        reltol = tol/tol0;
2517
2518    }
```

```
2519     if (iter >= maxiter)
2520    {
2521       cout<<"\n Number of iteration exceeds maximum. tol:  "<<tol;
2522       cout<< "\nelem: "<<CurrElementNumber()<<"\nIP: "<<CurrIP();
2523       ExceptionT::GeneralFail("ET_multi::Compute_a", "number of
               iteration exceeds maximum");
2524    }
2525 }
2526
2527 void ET_multi::Compute_le(const ArrayT<dSymMatrixT>& C_vn, ArrayT<
       dSymMatrixT>& C_v, const dArrayT& Tfk_n, dArrayT& Tfk, const ArrayT<
       dSymMatrixT>& C_Nvn, ArrayT<dSymMatrixT>& C_Nv, double& heat)
2528 {
2529
2530     double ctol = 1.00e-7;
2531    int maxiter = 100;
2532
2533    /*time step*/
2534    const double dt = fFSMatSupport->TimeStep();
2535
2536    /*temperature and temperature step*/
2537     const double Tn = Compute_Temperature();
2538     const double Tp = Compute_Temperature_last();
2539
2540     const dMatrixT& F_last = F_total_last();
2541     if (NumSD() == 2)
2542    {
2543       fF3D_last[0] = F_last[0];
2544       fF3D_last[1] = F_last[1];
2545       fF3D_last[2] = 0.0;
2546
2547       fF3D_last[3] = F_last[2];
2548       fF3D_last[4] = F_last[3];
2549       fF3D_last[5] = 0.0;
2550
2551       fF3D_last[6] = 0.0;
2552       fF3D_last[7] = 0.0;
2553       fF3D_last[8] = 1.0;
2554    }
2555    else fF3D_last = F_last;
2556
2557     fb_last.MultAAT(fF3D_last);
2558    fSpectralDecompSpat.SpectralDecomp_Jacobi(fb_last, false);
2559    fEigs_last = fSpectralDecompSpat.Eigenvalues();
2560
2561     double l0_last = fEigs_last[0];
2562     double l1_last = fEigs_last[1];
2563     double l2_last = fEigs_last[2];
2564     double Jp = sqrt(l0_last*l1_last*l2_last);
2565
2566     const dMatrixT& F = F_total();
2567    if (NumSD() == 2)
2568    {
2569       fF3D[0] = F[0];
2570       fF3D[1] = F[1];
2571       fF3D[2] = 0.0;
```

```
2572
2573        fF3D[3] = F[2];
2574        fF3D[4] = F[3];
2575        fF3D[5] = 0.0;
2576
2577        fF3D[6] = 0.0;
2578        fF3D[7] = 0.0;
2579        fF3D[8] = 1.0;
2580     }
2581     else fF3D = F;
2582
2583      fb.MultAAT(fF3D);
2584     fSpectralDecompb.SpectralDecomp_Jacobi(fb, false);
2585     fEigs = fSpectralDecompb.Eigenvalues();
2586      fEigMatb = fSpectralDecompb.Eigenmatrix();
2587      double J = sqrt(fEigs.Product());
2588     double iJ = 1.0/J;
2589
2590      dMatrixT U;
2591      U.Dimension(3);
2592
2593      fEigs_dev = fEigs;
2594     fEigs_dev *= pow(J, -2.0*third);
2595     fPot[0]->DevStress(fEigs_dev, ftau_EQ);
2596
2597      double l0 = fEigs[0];
2598      double l1 = fEigs[1];
2599      double l2 = fEigs[2];
2600
2601      double depsilon0=(0.5*log(l0)-0.5*log(l0_last));
2602      double depsilon1=(0.5*log(l1)-0.5*log(l1_last));
2603      double depsilon2=(0.5*log(l2)-0.5*log(l2_last));
2604
2605     /*calc trial solution l_tr, smag, and platicw*/
2606      double* pltr = fl_tr.Pointer();
2607     double* ple = fle.Pointer();
2608      double* pstress = fstressk.Pointer();
2609      double* pbackstress = fbackstressk.Pointer();
2610      double* pNstress = fNstressk.Pointer();
2611      double* pflowstress = fflowstressk.Pointer();
2612      double* pavectr = favec_tr.Pointer();
2613      double* pavec = favec.Pointer();
2614
2615      ftau = 0.0;
2616      fsback = 0.0;
2617     /*calculate trial state*/
2618     for (int i = 0; i < fNumS; i++)
2619      {
2620        /*calc trial elastic stretch*/
2621        fInverse.Inverse(C_vn[i]);
2622        fbe.MultQBQT(fF3D, fInverse);
2623        fSpectralDecompSpat.SpectralDecomp_Jacobi(fbe, false);
2624          fEigvec2=fSpectralDecompSpat.Eigenmatrix();
2625        fEigs_e = fSpectralDecompSpat.Eigenvalues();
2626          Reorder(fEigs_e, fEigs, fEigvec2, fEigMatb);
2627        *pltr++ = fEigs_e[0];
```

```
2628        *pltr++ = fEigs_e[1];
2629        *pltr++ = fEigs_e[2];
2630        /*initial condition be = btr*/
2631        *ple++ = fEigs_e[0];
2632        *ple++ = fEigs_e[1];
2633        *ple++ = fEigs_e[2];
2634          /*calculate total neq stress*/
2635        double Je = sqrt(fEigs_e.Product());
2636        fEigs_dev = fEigs_e;
2637        fEigs_dev *= pow(Je,-2.0*third);
2638        fPot[1]->DevStress(fEigs_dev, ftau_NEQ);
2639        ftau_NEQ *=fdmu[i];
2640          *pstress++ =ftau_NEQ[0];
2641          *pstress++ =ftau_NEQ[1];
2642          *pstress++ =ftau_NEQ[2];
2643
2644          ftau+= ftau_NEQ;
2645          //Get rotation tensor fR
2646          fSpectralDecompTest.PolarDecomp(fF3D, fR, fU, false);
2647          fSpectralDecompTest.SpectralDecomp_Jacobi(fU, false);
2648          //Calculate viscous right strech tensor Uv
2649          fSpectralDecompRef.SpectralDecomp_Jacobi(C_vn[i],false);
2650          fEigs_cvn = fSpectralDecompRef.Eigenvalues();
2651          fEigs_cvn[0]=sqrt(fEigs_cvn[0]);
2652          fEigs_cvn[1]=sqrt(fEigs_cvn[1]);
2653          fEigs_cvn[2]=sqrt(fEigs_cvn[2]);
2654          fUv=fSpectralDecompRef.EigsToRank2(fEigs_cvn);
2655          fUv.ToMatrix(U);
2656          //calculate Fv = R Uv
2657          fFv3D.MultAB(fR,U);
2658          //Calculate bNe_tr = Fv CNvn^-1 FvT
2659          fInverse.Inverse(C_Nvn[i]);
2660          fa.MultQBQT(fFv3D, fInverse);
2661
2662          fSpectralDecompSpat.SpectralDecomp_Jacobi(fa, false);
2663          fEigs_a = fSpectralDecompSpat.Eigenvalues();
2664          fEigvec1 = fSpectralDecompSpat.Eigenmatrix();
2665
2666          Reorder(fEigs_a, fEigs,fEigvec1, fEigMatb);
2667
2668          //write to a vector
2669          *pavec++ = fEigs_a[0];
2670          *pavec++ = fEigs_a[1];
2671          *pavec++ = fEigs_a[2];
2672
2673          *pavectr++ = fEigs_a[0];
2674          *pavectr++ = fEigs_a[1];
2675          *pavectr++ = fEigs_a[2];
2676
2677          //calculate trial backstress
2678        double Ja = sqrt(fEigs_a.Product());
2679        fEigs_a_dev = fEigs_a;
2680        fEigs_a_dev *= pow(Ja,-2.0*third);
2681
2682          fPot[2]->DevStress(fEigs_a_dev, ftau_BACK, fT0);
2683        ftau_BACK *= fHNratio*fdmu[i];
```

```
2684
2685        *pbackstress++ =ftau_BACK[0];
2686        *pbackstress++ =ftau_BACK[1];
2687        *pbackstress++ =ftau_BACK[2];
2688
2689        fsback += ftau_BACK;
2690    }
2691
2692     /*compute a*/
2693     Compute_a(fEigs_last, fEigs,fl_tr, ftau_EQ, fle, fstressk,
            fbackstressk, Tfk_n, Tfk, favec, favec_tr);
2694
2695     pavec = favec.Pointer();
2696
2697     double ScTn = 0.0;
2698     double prefactor = 1.0;
2699     double prefactor2 = 1.0;
2700     double Tfn = 0.0;
2701     double Tfp = 0.0;
2702     double plasticw = 0.0; //initially zero because epe-eptr = 0.0;
2703     double backw = 0.0;
2704
2705    for(int k=0; k<fNumR; k++)
2706    {
2707       double phi=fdalpha[k];
2708         fdeltac[k]=capacityc(Tfk[k]);
2709         ffrac1[k]=1.0/fdeltac[k];
2710         ffrac2[k]=fafrac/fdeltac[k];
2711         ffrac3[k]=fbfrac/fdeltac[k];
2712         Tfn += (Tfk[k]*phi);
2713         Tfp += (Tfk_n[k]*phi);
2714         ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-fT2);
2715         prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
2716         prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
2717         flatenth[k] =  Tfk[k]/fT0*(ftau_EQ[0]*depsilon0+ftau_EQ[1]*
               depsilon1+ftau_EQ[2]*depsilon2);
2718     }
2719
2720    fEigs_dev = fEigs;
2721    fEigs_dev *= pow(J, -2.0*third);
2722     double energy = fPot[0]->Energy(fEigs_dev, J);
2723     energy -= fPot[0]->MeanEnergy(J);
2724     ScTn -= energy/fT0;
2725
2726     fzeta=0.0;
2727     fNstress=0.0;
2728     pstress = fstressk.Pointer();
2729     pbackstress = fbackstressk.Pointer();
2730     pNstress = fNstressk.Pointer();
2731     pltr = fl_tr.Pointer();
2732     ple = fle.Pointer();
2733     pavectr = favec_tr.Pointer();
2734     pavec = favec.Pointer();
2735     for (int i = 0; i < fNumS; i++)
2736    {
2737         ftau_NEQ[0] =  *pstress++;
```

```
2738            ftau_NEQ[1] =  *pstress++;
2739            ftau_NEQ[2] =  *pstress++;
2740
2741            ftau_BACK[0] = *pbackstress++;
2742            ftau_BACK[1] = *pbackstress++;
2743            ftau_BACK[2] = *pbackstress++;
2744
2745            ftau_FLOW[0] = prefactor*ftau_NEQ[0]-prefactor2*ftau_BACK[0];
2746            ftau_FLOW[1] = prefactor*ftau_NEQ[1]-prefactor2*ftau_BACK[1];
2747            ftau_FLOW[2] = prefactor*ftau_NEQ[2]-prefactor2*ftau_BACK[2];
2748
2749            *pflowstress++ = ftau_FLOW[0];
2750            *pflowstress++ = ftau_FLOW[1];
2751            *pflowstress++ = ftau_FLOW[2];
2752
2753            fzeta += ftau_FLOW;
2754
2755            double epse0 = 0.5*log(*ple++);
2756            double epse1 = 0.5*log(*ple++);
2757            double epse2 = 0.5*log(*ple++);
2758
2759            double epstr0 = 0.5*log(*pltr++);
2760            double epstr1 = 0.5*log(*pltr++);
2761            double epstr2 = 0.5*log(*pltr++);
2762
2763            double epse0N = 0.5*log(*pavec++);
2764            double epse1N = 0.5*log(*pavec++);
2765            double epse2N = 0.5*log(*pavec++);
2766
2767            double epstr0N = 0.5*log(*pavectr++);
2768            double epstr1N = 0.5*log(*pavectr++);
2769            double epstr2N = 0.5*log(*pavectr++);
2770
2771            ftau_N = ftau_BACK;
2772            ftau_N *= prefactor2;
2773
2774            *pNstress++ = ftau_N[0];
2775            *pNstress++ = ftau_N[1];
2776            *pNstress++ = ftau_N[2];
2777
2778            fNstress += ftau_N;
2779
2780            double plasticwk = -(ftau_NEQ[0]*(epse0-epstr0)+ ftau_NEQ[1]*(
                    epse1-epstr1) + ftau_NEQ[2]*(epse2-epstr2));
2781            plasticw += plasticwk;
2782
2783            double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK[1]*(
                    epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
2784            backw += backwk;
2785        }
2786
2787        double zeta0n = fzeta[0];
2788        double zeta1n = fzeta[1];
2789        double zeta2n = fzeta[2];
2790
2791        double smag = sqrt(0.5*(zeta0n*zeta0n+zeta1n*zeta1n+zeta2n*zeta2n));
```

```
2792
2793
2794     double N0n = fNstress[0];
2795     double N1n = fNstress[1];
2796     double N2n = fNstress[2];
2797
2798     double sbmag = sqrt(0.5*(N0n*N0n+N1n*N1n+N2n*N2n));
2799
2800    /*calc viscosity functions*/
2801    double ietabar = StressRelaxationFunc(Tn, smag);
2802     double itaubar = StructuralRelaxationFunc(Tn, ScTn);
2803
2804     double ietabarN = ReptationRelaxationFuncS(Tn, sbmag);
2805     double itaubarN = ReptationRelaxationFuncT(Tn);
2806
2807     double lm0n=sqrt(fEigs[0]);
2808     double lm1n=sqrt(fEigs[1]);
2809     double lm2n=sqrt(fEigs[2]);
2810     double lmcn=sqrt(lm0n*lm0n+lm1n*lm1n+lm2n*lm2n);
2811
2812     double alpha0n=alphakf(lm0n,lmcn);
2813     double alpha1n=alphakf(lm1n,lmcn);
2814     double alpha2n=alphakf(lm2n,lmcn);
2815
2816     double alphamax=fmax(alpha0n,alpha1n);
2817     alphamax=fmax(alphamax,alpha2n);
2818     double alphan=0.5*pi-alphamax;
2819
2820     double alphafuncn=falpha(alphan);
2821
2822     double dalphadl0n,dalphadl1n,dalphadl2n;
2823
2824     if (alphamax == alpha0n)
2825    {
2826        dalphadl0n=sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2827          dalphadl1n=-lm0n*lm1n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2828          dalphadl2n=-lm0n*lm2n/sqrt(lm1n*lm1n+lm2n*lm2n)/lmcn/lmcn;
2829    }
2830    else if (alphamax == alpha1n)
2831    {
2832        dalphadl0n=-lm0n*lm1n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2833          dalphadl1n=sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2834          dalphadl2n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm2n*lm2n)/lmcn/lmcn;
2835    }
2836    else if (alphamax == alpha2n)
2837    {
2838        dalphadl0n=-lm0n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2839          dalphadl1n=-lm1n*lm2n/sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2840          dalphadl2n=sqrt(lm0n*lm0n+lm1n*lm1n)/lmcn/lmcn;
2841    }
2842
2843    /*calc residual*/
2844    /*re-assign pointers*/
2845    pltr = fl_tr.Pointer();
2846    ple = fle.Pointer();
2847     pstress = fstressk.Pointer();
```

```
2848     pbackstress = fbackstressk.Pointer();
2849     pflowstress = fflowstressk.Pointer();
2850     pavectr = favec_tr.Pointer();
2851     pavec = favec.Pointer();
2852 
2853    double* pr = fRes.Pointer();
2854    fRes = 0.0;
2855    double r0, r1, r2, tol;
2856    double temp = 0.0;
2857    for (int k = 0; k < fNumS; k++)
2858    {
2859       double epstr0 = 0.5*log(*pltr++);
2860       double epstr1 = 0.5*log(*pltr++);
2861       double epstr2 = 0.5*log(*pltr++);
2862 
2863       /*calculate neq stress tauk*/
2864       fEigs_dev[0] = *ple++;
2865       fEigs_dev[1] = *ple++;
2866       fEigs_dev[2] = *ple++;
2867       double epse0 = 0.5*log(fEigs_dev[0]);
2868       double epse1 = 0.5*log(fEigs_dev[1]);
2869       double epse2 = 0.5*log(fEigs_dev[2]);
2870 
2871         ftau_FLOW[0] = *pflowstress++;
2872         ftau_FLOW[1] = *pflowstress++;
2873         ftau_FLOW[2] = *pflowstress++;
2874 
2875         double s0 = ftau_FLOW[0];
2876         double s1 = ftau_FLOW[1];
2877         double s2 = ftau_FLOW[2];
2878 
2879       /*viscosity*/
2880         double muk=fdmu[k];
2881         double itauSk = itaubar/ftimesS[k];
2882         double ietaS0k = itauSk/muk;
2883         double ietaSk = ietaS0k*ietabar;
2884       /*calc residual*/
2885         r0 = epse0 - epstr0 + 0.5*dt*ietaSk*s0;
2886         r1 = epse1 - epstr1 + 0.5*dt*ietaSk*s1;
2887         r2 = epse2 - epstr2 + 0.5*dt*ietaSk*s2;
2888 
2889       temp += r0*r0 + r1*r1+ r2*r2;
2890       *pr++ = r0;
2891       *pr++ = r1;
2892       *pr++ = r2;
2893     }
2894    tol = sqrt(temp);
2895    double tol0 = tol;
2896    double reltol =tol0;
2897    int iter = 0;
2898 
2899     while (tol > ctol  && iter < maxiter)
2900    {
2901       iter++;
2902         fKAB2 = 0.0;
2903       fKAB = 0.0;
```

```
2904          fKrTe = 0.0;
2905          fKTer = 0.0;
2906          fKaAB2 = 0.0;
2907        fKaAB = 0.0;
2908          fKaTe = 0.0;
2909          fKTea = 0.0;
2910        fKdel = 0.0;
2911          fKar=0.0;
2912          fKra=0.0;
2913          fK11=0.0;
2914          fK12=0.0;
2915          fK21=0.0;
2916          fK22=0.0;
2917          fK11temp=0.0;
2918          fK12temp=0.0;
2919          fK21temp=0.0;
2920          fK22temp=0.0;
2921        ple = fle.Pointer();
2922          pltr = fl_tr.Pointer();
2923          pstress = fstressk.Pointer();
2924          pbackstress = fbackstressk.Pointer();
2925          pflowstress = fflowstressk.Pointer();
2926          pavectr = favec_tr.Pointer();
2927          pavec = favec.Pointer();
2928
2929          double DetabarDs = 0.0;
2930          if(smag > kSmall)
2931          {
2932              double x = (fQS*smag)/(Tn);
2933              DetabarDs = (x*cosh(x) - sinh(x))/x/smag;
2934          }
2935
2936          double DetabarDsN = 0.0;
2937          if(sbmag > kSmall)
2938          {
2939              double x = (fQSb*sbmag)/(Tn);
2940              DetabarDsN = (x*cosh(x) - sinh(x))/x/sbmag;
2941          }
2942
2943        for(int k=0; k<fNumS; k++)
2944        {
2945              double epstr0 = 0.5*log(*pltr++);
2946              double epstr1 = 0.5*log(*pltr++);
2947              double epstr2 = 0.5*log(*pltr++);
2948
2949           fEigs_dev[0] = *ple++;
2950           fEigs_dev[1] = *ple++;
2951           fEigs_dev[2] = *ple++;
2952
2953              double epse0 = 0.5*log(fEigs_dev[0]);
2954              double epse1 = 0.5*log(fEigs_dev[1]);
2955              double epse2 = 0.5*log(fEigs_dev[2]);
2956
2957              /*calculate neq stress tauk*/
2958           double Je = sqrt(fEigs_dev[0]*fEigs_dev[1]*fEigs_dev[2]);
2959           fEigs_dev *= pow(Je,-2.0*third);
```

```
2960            fPot[1]->DevMod(fEigs_dev,fDtauDe_NEQ);
2961
2962          /*moduli*/
2963          /*viscosity and its derivatives*/
2964          double muk=fdmu[k];
2965             double itauSk = itaubar/ftimesS[k];
2966          double ietaS0k = itauSk/muk;
2967          double ietaSk = ietaS0k*ietabar;
2968
2969             double mukN=fHNratio*fdmu[k];
2970             double itauSkN = itaubarN/(ftimesS[k]*fshift);
2971             double ietaS0kN = itauSkN/mukN;
2972             double ietaSkN = ietaS0kN*ietabarN;
2973
2974          double dk00 = muk*iJ*fDtauDe_NEQ(0,0);
2975          double dk11 = muk*iJ*fDtauDe_NEQ(1,1);
2976          double dk22 = muk*iJ*fDtauDe_NEQ(2,2);
2977          double dk12 = muk*iJ*fDtauDe_NEQ(1,2);
2978          double dk02 = muk*iJ*fDtauDe_NEQ(0,2);
2979          double dk01 = muk*iJ*fDtauDe_NEQ(0,1);
2980
2981             double a0k = *pavec++;
2982             double a1k = *pavec++;
2983             double a2k = *pavec++;
2984
2985             double epse0N = 0.5*log(a0k);
2986             double epse1N = 0.5*log(a1k);
2987             double epse2N = 0.5*log(a2k);
2988
2989             double epstr0N = 0.5*log(*pavectr++);
2990             double epstr1N = 0.5*log(*pavectr++);
2991             double epstr2N = 0.5*log(*pavectr++);
2992
2993             fEigs_a[0] = a0k;
2994          fEigs_a[1] = a1k;
2995          fEigs_a[2] = a2k;
2996
2997             double Ja = sqrt(fEigs_a.Product());
2998             fEigs_a_dev = fEigs_a;
2999             fEigs_a_dev *= pow(Ja,-2.0*third);
3000             fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
3001          fDtauDe_BACK *= fHNratio*fdmu[k];
3002
3003             double dN00k = prefactor2*fDtauDe_BACK(0,0);
3004             double dN01k = prefactor2*fDtauDe_BACK(0,1);
3005             double dN02k = prefactor2*fDtauDe_BACK(0,2);
3006             double dN10k = prefactor2*fDtauDe_BACK(0,1);
3007             double dN11k = prefactor2*fDtauDe_BACK(1,1);
3008             double dN12k = prefactor2*fDtauDe_BACK(1,2);
3009             double dN20k = prefactor2*fDtauDe_BACK(0,2);
3010             double dN21k = prefactor2*fDtauDe_BACK(1,2);
3011             double dN22k = prefactor2*fDtauDe_BACK(2,2);
3012
3013             double tauk0 =  *pstress++;
3014             double tauk1 =  *pstress++;
3015             double tauk2 =  *pstress++;
```

```
3016            double Mk0n = (tauk0)*prefactor;
3017          double Mk1n = (tauk1)*prefactor;
3018          double Mk2n = (tauk2)*prefactor;
3019
3020            double taukback0 =  *pbackstress++;
3021            double taukback1 =  *pbackstress++;
3022            double taukback2 =  *pbackstress++;
3023            double Nk0n = (taukback0)*prefactor2;
3024            double Nk1n = (taukback1)*prefactor2;
3025            double Nk2n = (taukback2)*prefactor2;
3026
3027            double zetak0 =  *pflowstress++;
3028            double zetak1 =  *pflowstress++;
3029            double zetak2 =  *pflowstress++;
3030
3031          /*stiffness matrix*/
3032          int a = k*3;
3033          fKaAB(a,a) = 1.0 +0.5*dt*ietaSkN/alphafuncn*dN00k;
3034          fKaAB(a,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN01k;
3035          fKaAB(a,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN02k;
3036          fKaAB(a+1,a) = 0.5*dt*ietaSkN/alphafuncn*dN10k;
3037          fKaAB(a+1,a+1) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN11k;
3038          fKaAB(a+1,a+2) = 0.5*dt*ietaSkN/alphafuncn*dN12k;
3039          fKaAB(a+2,a) = 0.5*dt*ietaSkN/alphafuncn*dN20k;
3040          fKaAB(a+2,a+1) = 0.5*dt*ietaSkN/alphafuncn*dN21k;
3041          fKaAB(a+2,a+2) = 1.0 + 0.5*dt*ietaSkN/alphafuncn*dN22k;
3042
3043            fKar(a,a) = 1.0;
3044          fKar(a,a+1) = 0.0;
3045          fKar(a,a+2) = 0.0;
3046          fKar(a+1,a) = 0.0;
3047          fKar(a+1,a+1) = 1.0;
3048          fKar(a+1,a+2) = 0.0;
3049          fKar(a+2,a) = 0.0;
3050          fKar(a+2,a+1) = 0.0;
3051          fKar(a+2,a+2) = 1.0;
3052
3053            fKra(a,a) = -0.5*dt*ietaSk*dN00k;
3054          fKra(a,a+1) = -0.5*dt*ietaSk*dN01k;
3055          fKra(a,a+2) = -0.5*dt*ietaSk*dN02k;
3056          fKra(a+1,a) = -0.5*dt*ietaSk*dN10k;
3057          fKra(a+1,a+1) = -0.5*dt*ietaSk*dN11k;
3058          fKra(a+1,a+2) = -0.5*dt*ietaSk*dN12k;
3059          fKra(a+2,a) = -0.5*dt*ietaSk*dN20k;
3060          fKra(a+2,a+1) = -0.5*dt*ietaSk*dN21k;
3061          fKra(a+2,a+2) = -0.5*dt*ietaSk*dN22k;
3062
3063          fKAB(a,a) = 1.0 + 0.5*dt*ietaSk*dk00*prefactor/iJ;
3064          fKAB(a,a+1) = 0.5*dt*ietaSk*dk01*prefactor/iJ;
3065          fKAB(a,a+2) = 0.5*dt*ietaSk*dk02*prefactor/iJ;
3066          fKAB(a+1,a) = 0.5*dt*ietaSk*dk01*prefactor/iJ;
3067          fKAB(a+1,a+1) = 1.0 + 0.5*dt*ietaSk*dk11*prefactor/iJ;
3068          fKAB(a+1,a+2) = 0.5*dt*ietaSk*dk12*prefactor/iJ;
3069          fKAB(a+2,a) = 0.5*dt*ietaSk*dk02*prefactor/iJ;
3070          fKAB(a+2,a+1) = 0.5*dt*ietaSk*dk12*prefactor/iJ;
3071          fKAB(a+2,a+2) = 1.0 + 0.5*dt*ietaSk*dk22*prefactor/iJ;
```

```
3072
3073            double *ple2 = fle.Pointer();
3074            double *paEigVec = favec.Pointer();
3075          for (int l=0; l<fNumS; l++)       /*evolution eqs. are coupled
                in this case because of s (or smag here)*/
3076          {
3077            /*moduli*/
3078            fEigs_dev[0] = *ple2++;
3079            fEigs_dev[1] = *ple2++;
3080            fEigs_dev[2] = *ple2++;
3081            double Je2 = sqrt(fEigs_dev[0]*fEigs_dev[1]*fEigs_dev[2]);
3082            fEigs_dev *= pow(Je2,-2.0*third);
3083            fPot[1]->DevMod(fEigs_dev,fDtauDe_NEQ);
3084            double dl00 = fdmu[l]*iJ*fDtauDe_NEQ(0,0);
3085            double dl11 = fdmu[l]*iJ*fDtauDe_NEQ(1,1);
3086            double dl22 = fdmu[l]*iJ*fDtauDe_NEQ(2,2);
3087            double dl12 = fdmu[l]*iJ*fDtauDe_NEQ(1,2);
3088            double dl02 = fdmu[l]*iJ*fDtauDe_NEQ(0,2);
3089            double dl01 = fdmu[l]*iJ*fDtauDe_NEQ(0,1);
3090
3091                double cl0 = 0.0;
3092                double cl1 = 0.0;
3093                double cl2 = 0.0;
3094
3095                double a0l = *paEigVec++;
3096                double a1l = *paEigVec++;
3097                double a2l = *paEigVec++;
3098
3099                fEigs_a[0] = a0l;
3100                fEigs_a[1] = a1l;
3101                fEigs_a[2] = a2l;
3102
3103                double Ja = sqrt(fEigs_a.Product());
3104                fEigs_a_dev = fEigs_a;
3105                fEigs_a_dev *= pow(Ja,-2.0*third);
3106                fPot[2]->DevMod(fEigs_a_dev,fDtauDe_BACK, fT0);
3107               fDtauDe_BACK *= fHNratio*fdmu[l];
3108
3109                double dN00l = prefactor2*fDtauDe_BACK(0,0);
3110                double dN01l = prefactor2*fDtauDe_BACK(0,1);
3111                double dN02l = prefactor2*fDtauDe_BACK(0,2);
3112                double dN10l = prefactor2*fDtauDe_BACK(0,1);
3113                double dN11l = prefactor2*fDtauDe_BACK(1,1);
3114                double dN12l = prefactor2*fDtauDe_BACK(1,2);
3115                double dN20l = prefactor2*fDtauDe_BACK(0,2);
3116                double dN21l = prefactor2*fDtauDe_BACK(1,2);
3117                double dN22l = prefactor2*fDtauDe_BACK(2,2);
3118
3119                double dl0 = 0.0;
3120                double dl1 = 0.0;
3121                double dl2 = 0.0;
3122
3123                double el0 = 0.0;
3124                double el1 = 0.0;
3125                double el2 = 0.0;
3126
```

```
3127                    if(smag >kSmall)
3128                    {
3129                        c10 = 0.5*(fzeta[0]*dl00 + fzeta[1]*dl01 + fzeta[2]*
                                dl02)*prefactor/iJ/smag;
3130                        c11 = 0.5*(fzeta[0]*dl01 + fzeta[1]*dl11 + fzeta[2]*
                                dl12)*prefactor/iJ/smag;
3131                        c12 = 0.5*(fzeta[0]*dl02 + fzeta[1]*dl12 + fzeta[2]*
                                dl22)*prefactor/iJ/smag;
3132
3133                        d10 = -0.5*(fzeta[0]*dN00l + fzeta[1]*dN10l + fzeta
                                [2]*dN20l)/smag;
3134                        d11 = -0.5*(fzeta[0]*dN01l + fzeta[1]*dN11l + fzeta
                                [2]*dN21l)/smag;
3135                        d12 = -0.5*(fzeta[0]*dN02l + fzeta[1]*dN12l + fzeta
                                [2]*dN22l)/smag;
3136                    }
3137
3138                    if(sbmag >kSmall)
3139                    {
3140                        e10 = 0.5*(fNstress[0]*dN00l + fNstress[1]*dN10l +
                                fNstress[2]*dN20l)/sbmag;
3141                        e11 = 0.5*(fNstress[0]*dN01l + fNstress[1]*dN11l +
                                fNstress[2]*dN21l)/sbmag;
3142                        e12 = 0.5*(fNstress[0]*dN02l + fNstress[1]*dN12l +
                                fNstress[2]*dN22l)/sbmag;
3143                    }
3144
3145                int b = 1*3;
3146                fKaAB(a,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*Nk0n
                        ;
3147                fKaAB(a,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk0n;
3148                fKaAB(a,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e12*
                        Nk0n;
3149                fKaAB(a+1,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*
                        Nk1n;
3150                fKaAB(a+1,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk1n;
3151                fKaAB(a+1,b+2)+= 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e12*
                        Nk1n;
3152                fKaAB(a+2,b) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e10*
                        Nk2n;
3153                fKaAB(a+2,b+1) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*e11*
                        Nk2n;
3154                    fKaAB(a+2,b+2) += 0.5*dt*ietaS0kN/alphafuncn*DetabarDsN*
                            e12*Nk2n;
3155
3156                    fKra(a,b) +=  0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak0;
3157                fKra(a,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak0;
3158                fKra(a,b+2) += 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak0;
3159                fKra(a+1,b) += 0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak1;
3160                fKra(a+1,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak1;
3161                fKra(a+1,b+2)+= 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak1;
3162                fKra(a+2,b) += 0.5*dt*ietaS0k*(DetabarDs*dl0)*zetak2;
3163                fKra(a+2,b+1) += 0.5*dt*ietaS0k*(DetabarDs*dl1)*zetak2;
3164                    fKra(a+2,b+2) += 0.5*dt*ietaS0k*(DetabarDs*dl2)*zetak2;
```

```
3165
3166            fKAB(a,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak0;
3167            fKAB(a,b+1) += 0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak0;
3168            fKAB(a,b+2) +=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak0;
3169            fKAB(a+1,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak1;
3170            fKAB(a+1,b+1) +=  0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak1;
3171            fKAB(a+1,b+2)+=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak1;
3172            fKAB(a+2,b) +=  0.5*dt*ietaS0k*(DetabarDs*cl0)*zetak2;
3173            fKAB(a+2,b+1) +=  0.5*dt*ietaS0k*(DetabarDs*cl1)*zetak2;
3174                fKAB(a+2,b+2) +=  0.5*dt*ietaS0k*(DetabarDs*cl2)*zetak2;
3175
3176            }
3177            for(int l=0; l<fNumR; l++)
3178            {
3179                fKTea(l,a) = ffrac3[l]*((dN00k*(epse0N-epstr0N)+dN10k*(
                        epse1N-epstr1N)+dN20k*(epse2N-epstr2N))/prefactor2+
                        taukback0);
3180                fKTea(l,a+1) = ffrac3[l]*((dN01k*(epse0N-epstr0N)+dN11k
                        *(epse1N-epstr1N)+dN21k*(epse2N-epstr2N))/prefactor2
                        +taukback1);
3181                fKTea(l,a+2) = ffrac3[l]*((dN02k*(epse0N-epstr0N)+dN12k
                        *(epse1N-epstr1N)+dN22k*(epse2N-epstr2N))/prefactor2
                        +taukback2);
3182
3183                double M0n = prefactor*ftau[0];
3184                double M1n = prefactor*ftau[1];
3185                double M2n = prefactor*ftau[2];
3186
3187                double N0n = prefactor2*fsback[0];
3188                double N1n = prefactor2*fsback[1];
3189                double N2n = prefactor2*fsback[2];
3190
3191                double zm = 0.5*(zeta0n*M0n+zeta1n*M1n+zeta2n*M2n);
3192                double zn = 0.5*(zeta0n*N0n+zeta1n*N1n+zeta2n*N2n);
3193
3194                double phi = fdalpha[l];
3195                double DsDTek = 0.0;
3196                if (smag>kSmall)
3197                    DsDTek = -phi*Tn*fafrac*zm/prefactor/smag/Tfk[l]/Tfk
                            [l]+phi*Tn*fbfrac*zn/prefactor2/smag/Tfk[l]/Tfk[
                            l];
3198
3199                double DsbDTek = 0.0;
3200                if (sbmag>kSmall)
3201                    DsbDTek = -phi*Tn*fbfrac*sbmag/prefactor2/Tfk[l]/Tfk
                            [l];
3202
3203                fKaTe(a,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk0n*DetabarDsN*
                        DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*phi*Tn/Tfk[
                        l]/Tfk[l]*taukback0;
3204                fKaTe(a+1,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk1n*
                        DetabarDsN*DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*
                        phi*Tn/Tfk[l]/Tfk[l]*taukback1;
3205                fKaTe(a+2,l) = 0.5*dt*ietaS0kN/alphafuncn*Nk2n*
                        DetabarDsN*DsbDTek-0.5*dt*ietaSkN/alphafuncn*fbfrac*
                        phi*Tn/Tfk[l]/Tfk[l]*taukback2;
```

```
3206
3207                fKTer(l,a) = ffrac2[l]*(dk00/iJ*(epse0-epstr0) + tauk0 +
                        dk01/iJ*(epse1-epstr1)+dk02/iJ*(epse2-epstr2))+
                        ffrac3[l]*taukback0;
3208                fKTer(l,a+1) = ffrac2[l]*(dk01/iJ*(epse0-epstr0) + tauk1
                        + dk11/iJ*(epse1-epstr1)+dk12/iJ*(epse2-epstr2))+
                        ffrac3[l]*taukback1;
3209                fKTer(l,a+2) = ffrac2[l]*(dk02/iJ*(epse0-epstr0) + tauk2
                        + dk12/iJ*(epse1-epstr1)+dk22/iJ*(epse2-epstr2))+
                        ffrac3[l]*taukback2;
3210
3211                fKrTe(a,l) =    0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*
                        fdeltac0/Tfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/
                        ietabar)*zetak0 +0.5*dt*ietaSk*(-Mk0n*phi*Tn*fafrac/
                        prefactor/Tfk[l]/Tfk[l]+Nk0n*phi*Tn*fbfrac/
                        prefactor2/Tfk[l]/Tfk[l]);
3212                fKrTe(a+1,l) = 0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*
                        fdeltac0/Tfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/
                        ietabar)*zetak1 +0.5*dt*ietaSk*(-Mk1n*phi*Tn*fafrac/
                        prefactor/Tfk[l]/Tfk[l]+Nk1n*phi*Tn*fbfrac/
                        prefactor2/Tfk[l]/Tfk[l]);
3213                fKrTe(a+2,l) = 0.5*dt*ietaSk*(fBB/Tn/ScTn/ScTn*(phi*
                        fdeltac0/Tfk[l]+fdeltac1*phi)+DetabarDs*DsDTek/
                        ietabar)*zetak2 +0.5*dt*ietaSk*(-Mk2n*phi*Tn*fafrac/
                        prefactor/Tfk[l]/Tfk[l]+Nk2n*phi*Tn*fbfrac/
                        prefactor2/Tfk[l]/Tfk[l]);
3214
3215            }
3216        }
3217        for(int k=0; k<fNumR; k++)
3218    {
3219         /*stiffness matrix*/
3220         double itauRk = itaubar/ftimesR[k];
3221            fKdel(k,k) = 1.0 + dt*itauRk-ffrac1[k]*flatenth[k]/Tfk[k]+
                fafrac*ffrac1[k]*ffrac1[k]*fdeltac1*plasticw+ffrac1[k]*
                ffrac1[k]*fdeltac1*flatenth[k]-fbfrac*ffrac1[k]*ffrac1[k
                ]*fdeltac1*backw;
3222         for (int l=0; l<fNumR; l++)
3223            {
3224                double phi=fdalpha[l];
3225                double DtaubarDTfk = -fBB/Tn/ScTn/ScTn*(phi*fdeltac0/Tfk
                    [l]+fdeltac1*phi);
3226            fKdel(k,l) += -dt*itauRk*(Tfk[k]-Tn)*DtaubarDTfk;
                        }
3227    }
3228
3229        fK11=fKaAB;
3230        fK12=fKar;
3231        fK21=fKTer;
3232        fK22=fKdel;
3233        fKdel.Inverse();
3234        fKaAB.Inverse();
3235        fK11temp.MultABC(fKaTe, fKdel, fKTea);
3236        fK12temp.MultABC(fKaTe, fKdel, fKTer);
3237        fK21temp.MultABC(fKTea, fKaAB, fKar);
3238        fK22temp.MultABC(fKTea, fKaAB, fKaTe);
```

```
3239          fK11.AddScaled(-1.0, fK11temp);
3240          fK12.AddScaled(-1.0, fK12temp);
3241          fK21.AddScaled(-1.0, fK21temp);
3242          fK22.AddScaled(-1.0, fK22temp);
3243          fK11.Inverse();
3244          fK22.Inverse();
3245
3246          fKaAB2.MultABC(fKra, fK11, fK12);
3247          fKAB2.MultABC(fKrTe, fK22, fK21);
3248          fKAB.AddScaled(-1.0, fKaAB2);
3249          fKAB.AddScaled(-1.0, fKAB2);
3250
3251          /*Solve  and update*/
3252        fKAB.LinearSolve(fRes);
3253          /*update Cv and be*/
3254        ple = fle.Pointer();
3255        pr = fRes.Pointer();
3256
3257        for (int k = 0; k < 3*fNumS; k++)
3258        {
3259           double dep = -(*pr++);
3260           *ple++ *= exp(2.0*dep);
3261        }
3262
3263          ple = fle.Pointer();
3264          pavectr = favec_tr.Pointer();
3265          pavec = favec.Pointer();
3266          pbackstress = fbackstressk.Pointer();
3267
3268          //update trial state for a=bNe
3269          for (int k = 0; k < fNumS; k++)
3270        {
3271              fEigs_e[0] = *ple++;
3272              fEigs_e[1] = *ple++;
3273              fEigs_e[2] = *ple++;
3274              fbe = fSpectralDecompb.EigsToRank2(fEigs_e);
3275              fInverse.Inverse(fbe);
3276              C_v[k].MultQTBQ(fF3D, fInverse);
3277              //Calculate Uv
3278              fSpectralDecompRef.SpectralDecomp_Jacobi(C_v[k],false);
3279              fEigs_cvn = fSpectralDecompRef.Eigenvalues();
3280              fEigs_cvn[0]=sqrt(fEigs_cvn[0]);
3281              fEigs_cvn[1]=sqrt(fEigs_cvn[1]);
3282              fEigs_cvn[2]=sqrt(fEigs_cvn[2]);
3283              fUv=fSpectralDecompRef.EigsToRank2(fEigs_cvn);
3284              fUv.ToMatrix(U);
3285              //Calculate Fv and update bNe (fa)
3286              fFv3D.MultAB(fR,U);
3287              fInverse.Inverse(C_Nvn[k]);
3288              fa.MultQBQT(fFv3D, fInverse);
3289              //Decompose and get favec
3290              fSpectralDecompSpat.SpectralDecomp_Jacobi(fa, false);
3291              fEigs_a = fSpectralDecompSpat.Eigenvalues();
3292              fEigvec1 = fSpectralDecompSpat.Eigenmatrix();
3293              Reorder(fEigs_a,fEigs, fEigvec1, fEigMatb);
3294
```

```
3295              *pavectr++ = fEigs_a[0];
3296              *pavectr++ = fEigs_a[1];
3297              *pavectr++ = fEigs_a[2];
3298          }
3299
3300          /*update stresses*/
3301          ple = fle.Pointer();
3302          pstress = fstressk.Pointer();
3303          ftau =0.0;
3304          for (int k = 0; k < fNumS; k++)
3305          {
3306              fEigs_e[0] = *ple++;
3307              fEigs_e[1] = *ple++;
3308              fEigs_e[2] = *ple++;
3309
3310              double Je = sqrt(fEigs_e.Product());
3311           fEigs_dev[0] = fEigs_e[0]*pow(Je,-2.0*third);
3312           fEigs_dev[1] = fEigs_e[1]*pow(Je,-2.0*third);
3313           fEigs_dev[2] = fEigs_e[2]*pow(Je,-2.0*third);
3314              fPot[1]->DevStress(fEigs_dev, ftau_NEQ);
3315              ftau_NEQ *=fdmu[k];
3316
3317              *pstress++ = ftau_NEQ[0];
3318              *pstress++ = ftau_NEQ[1];
3319              *pstress++ = ftau_NEQ[2];
3320
3321              ftau+= ftau_NEQ;
3322          }
3323
3324          /*compute a*/
3325         Compute_a(fEigs_last, fEigs,fl_tr, ftau_EQ, fle, fstressk,
                fbackstressk, Tfk_n, Tfk, favec, favec_tr);
3326
3327          double ScTn = 0.0;
3328          double prefactor = 1.0;
3329          double prefactor2 = 1.0;
3330          double Tfn = 0.0;
3331          double Tfp = 0.0;
3332          double plasticw = 0.0;
3333          double backw = 0.0;
3334
3335          for(int k=0; k<fNumR; k++)
3336          {
3337              double phi=fdalpha[k];
3338              fdeltac[k]=capacityc(Tfk[k]);
3339              ffrac1[k]=1.0/fdeltac[k];
3340              ffrac2[k]=fafrac/fdeltac[k];
3341              ffrac3[k]=fbfrac/fdeltac[k];
3342              Tfn += (Tfk[k]*phi);
3343              Tfp += (Tfk_n[k]*phi);
3344              ScTn += fdeltac0*phi*log(Tfk[k]/fT2)+fdeltac1*phi*(Tfk[k]-
                    fT2);
3345              prefactor += phi*fafrac*(Tn/Tfk[k]-1.0);
3346              prefactor2 += phi*fbfrac*(Tn/Tfk[k]-1.0);
3347              flatenth[k] =  Tfk[k]/fT0*(ftau_EQ[0]*depsilon0+ftau_EQ[1]*
                    depsilon1+ftau_EQ[2]*depsilon2);
```

```
3348            }
3349            fEigs_dev = fEigs;
3350            fEigs_dev *= pow(J, -2.0*third);
3351            energy = fPot[0]->Energy(fEigs_dev, J);
3352            energy -= fPot[0]->MeanEnergy(J);
3353            ScTn -= energy/fT0;
3354
3355            /*update pavec and back stress*/
3356            pavec = favec.Pointer();
3357            pbackstress = fbackstressk.Pointer();
3358            fsback=0.0;
3359            for (int i = 0; i < fNumS; i++)
3360            {
3361                fEigs_a[0] = *pavec++;
3362                fEigs_a[1] = *pavec++;
3363                fEigs_a[2] = *pavec++;
3364
3365                double Ja = sqrt(fEigs_a.Product());
3366                fEigs_a_dev = fEigs_a;
3367                fEigs_a_dev *= pow(Ja,-2.0*third);
3368
3369                fPot[2]->DevStress(fEigs_a_dev, ftau_BACK, fT0);
3370                ftau_BACK *= fHNratio*fdmu[i];
3371
3372                *pbackstress++ =ftau_BACK[0];
3373                *pbackstress++ =ftau_BACK[1];
3374                *pbackstress++ =ftau_BACK[2];
3375
3376                fsback += ftau_BACK;
3377            }
3378
3379            fzeta=0.0;
3380            fNstress=0.0;
3381            pstress = fstressk.Pointer();
3382            pbackstress = fbackstressk.Pointer();
3383            pNstress = fNstressk.Pointer();
3384            pflowstress = fflowstressk.Pointer();
3385            pltr = fl_tr.Pointer();
3386            ple = fle.Pointer();
3387            pavectr = favec_tr.Pointer();
3388            pavec = favec.Pointer();
3389            for (int i = 0; i < fNumS; i++)
3390            {
3391                ftau_NEQ[0] =  *pstress++;
3392                ftau_NEQ[1] =  *pstress++;
3393                ftau_NEQ[2] =  *pstress++;
3394
3395                ftau_BACK[0] = *pbackstress++;
3396                ftau_BACK[1] = *pbackstress++;
3397                ftau_BACK[2] = *pbackstress++;
3398
3399                ftau_FLOW[0] = prefactor*ftau_NEQ[0]-prefactor2*ftau_BACK
                        [0];
3400                ftau_FLOW[1] = prefactor*ftau_NEQ[1]-prefactor2*ftau_BACK
                        [1];
3401                ftau_FLOW[2] = prefactor*ftau_NEQ[2]-prefactor2*ftau_BACK
```

```
3402                [2];
3403
3404            *pflowstress++ = ftau_FLOW[0];
3405            *pflowstress++ = ftau_FLOW[1];
3406            *pflowstress++ = ftau_FLOW[2];
3407
3408            fzeta += ftau_FLOW;
3409
3410            double epse0 = 0.5*log(*ple++);
3411            double epse1 = 0.5*log(*ple++);
3412            double epse2 = 0.5*log(*ple++);
3413
3414            double epstr0 = 0.5*log(*pltr++);
3415            double epstr1 = 0.5*log(*pltr++);
3416            double epstr2 = 0.5*log(*pltr++);
3417
3418            double epse0N = 0.5*log(*pavec++);
3419            double epse1N = 0.5*log(*pavec++);
3420            double epse2N = 0.5*log(*pavec++);
3421
3422            double epstr0N = 0.5*log(*pavectr++);
3423            double epstr1N = 0.5*log(*pavectr++);
3424            double epstr2N = 0.5*log(*pavectr++);
3425
3426            ftau_N = ftau_BACK;
3427            ftau_N *= prefactor2;
3428
3429            *pNstress++ = ftau_N[0];
3430            *pNstress++ = ftau_N[1];
3431            *pNstress++ = ftau_N[2];
3432
3433            fNstress += ftau_N;
3434
3435            double plasticwk = -(ftau_NEQ[0]*(epse0-epstr0)+ ftau_NEQ
                   [1]*(epse1-epstr1) + ftau_NEQ[2]*(epse2-epstr2));
3436            plasticw += plasticwk;
3437
3438            double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK
                   [1]*(epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
3439            backw += backwk;
3440        }
3441
3442      /*update  smag*/
3443        double zeta0n = fzeta[0];
3444        double zeta1n = fzeta[1];
3445        double zeta2n = fzeta[2];
3446
3447        double smag = sqrt(0.5*(zeta0n*zeta0n+zeta1n*zeta1n+zeta2n*
               zeta2n));
3448
3449
3450        double N0n = fNstress[0];
3451        double N1n = fNstress[1];
3452        double N2n = fNstress[2];
3453
3454        double sbmag = sqrt(0.5*(N0n*N0n+N1n*N1n+N2n*N2n));
```

```
3454
3455        /*update viscosity*/
3456          ietabar = StressRelaxationFunc(Tn, smag);
3457          itaubar = StructuralRelaxationFunc(Tn, ScTn);
3458
3459          ietabarN = ReptationRelaxationFuncS(Tn, sbmag);
3460          itaubarN = ReptationRelaxationFuncT(Tn);
3461
3462          /*calc residual*/
3463          /*re-assign pointers*/
3464          pltr = fl_tr.Pointer();
3465          ple = fle.Pointer();
3466          pstress = fstressk.Pointer();
3467          pbackstress = fbackstressk.Pointer();
3468          pflowstress = fflowstressk.Pointer();
3469          pavectr = favec_tr.Pointer();
3470          pavec = favec.Pointer();
3471
3472          pr = fRes.Pointer();
3473          temp = 0.0;
3474          for (int k = 0; k < fNumS; k++)
3475          {
3476              double epstr0 = 0.5*log(*pltr++);
3477              double epstr1 = 0.5*log(*pltr++);
3478              double epstr2 = 0.5*log(*pltr++);
3479
3480              /*calculate neq stress tauk*/
3481              fEigs_dev[0] = *ple++;
3482              fEigs_dev[1] = *ple++;
3483              fEigs_dev[2] = *ple++;
3484              double epse0 = 0.5*log(fEigs_dev[0]);
3485              double epse1 = 0.5*log(fEigs_dev[1]);
3486              double epse2 = 0.5*log(fEigs_dev[2]);
3487
3488              ftau_FLOW[0] = *pflowstress++;
3489              ftau_FLOW[1] = *pflowstress++;
3490              ftau_FLOW[2] = *pflowstress++;
3491
3492              double s0 = ftau_FLOW[0];
3493              double s1 = ftau_FLOW[1];
3494              double s2 = ftau_FLOW[2];
3495
3496              /*viscosity*/
3497              double muk=fdmu[k];
3498              double itauSk = itaubar/ftimesS[k];
3499              double ietaS0k = itauSk/muk;
3500              double ietaSk = ietaS0k*ietabar;
3501              /*calc residual*/
3502              r0 = epse0 - epstr0 + 0.5*dt*ietaSk*s0;
3503              r1 = epse1 - epstr1 + 0.5*dt*ietaSk*s1;
3504              r2 = epse2 - epstr2 + 0.5*dt*ietaSk*s2;
3505
3506              temp += r0*r0 + r1*r1+ r2*r2;
3507              *pr++ = r0;
3508              *pr++ = r1;
3509              *pr++ = r2;
```

```
3510            }
3511
3512            tol = sqrt(temp);
3513            reltol = tol/tol0;
3514        }
3515         if (iter >= maxiter)
3516        {
3517           cout<<"\n Number of iteration exceeds maximum. tol:  "<<tol;
3518           cout<< "\nelem: "<<CurrElementNumber()<<"\nIP: "<<CurrIP();
3519           ExceptionT::GeneralFail("ET_multi::Compute_le", "number of
                  iteration exceeds maximum");
3520        }
3521         if (isnan(tol))
3522         {
3523            cout<< "\nelem: "<<CurrElementNumber()<<"\nIP: "<<CurrIP();
3524            ExceptionT::GeneralFail("ET_multi::Compute_le", "nan");
3525         }
3526
3527        /*update Cv &CNv with converged solution*/
3528        ple = fle.Pointer();
3529         pavec = favec.Pointer();
3530        for (int k = 0; k < fNumS; k++)
3531        {
3532            fEigs_a[0] = *pavec++;
3533            fEigs_a[1] = *pavec++;
3534            fEigs_a[2] = *pavec++;
3535            fa=fSpectralDecompb.EigsToRank2(fEigs_a);
3536            fa.Inverse();
3537
3538            fSpectralDecompRef.SpectralDecomp_Jacobi(C_v[k],false);
3539            fEigs_cvn = fSpectralDecompRef.Eigenvalues();
3540            fEigs_cvn[0]=sqrt(fEigs_cvn[0]);fEigs_cvn[1]=sqrt(fEigs_cvn[1]);
                  fEigs_cvn[2]=sqrt(fEigs_cvn[2]);
3541            fU=fSpectralDecompRef.EigsToRank2(fEigs_cvn);
3542
3543            dMatrixT U; U.Dimension(3);
3544            fU.ToMatrix(U);
3545            fFv3D.MultAB(fR,U);
3546
3547            C_Nv[k].MultQTBQ(fFv3D,fa);
3548        }
3549
3550         /*calculate incremental heat*/
3551         const double rho = Density();
3552         double structural=0.0;
3553         double coeff_latentheat=0.0;
3554         for (int k = 0; k < fNumS; k++)
3555         {
3556            fdeltac[k]=capacityc(Tfk[k]);
3557            structural += rho*fdeltac[k]*fdalpha[k]*(Tfk[k]-Tfk_n[k]);
3558            coeff_latentheat += (Tfk[k]/fT0)*fdalpha[k];
3559         }
3560         /*add contributions from structural relaxation and latent heat*/
3561         heat = structural - coeff_latentheat*(ftau_EQ[0]*depsilon0+ftau_EQ
                  [1]*depsilon1+ftau_EQ[2]*depsilon2);
3562
```

```
3563     Tfn = 0.0;
3564     Tfp = 0.0;
3565     for(int k=0; k<fNumR; k++)
3566     {
3567         double phi=fdalpha[k];
3568         Tfn += (Tfk[k]*phi);
3569         Tfp += (Tfk_n[k]*phi);
3570     }
3571 
3572     backw=0.0;
3573     plasticw = 0.0;
3574     ple = fle.Pointer();
3575     pltr = fl_tr.Pointer();
3576     pstress = fstressk.Pointer();
3577     pbackstress = fbackstressk.Pointer();
3578     pavectr = favec_tr.Pointer();
3579     pavec = favec.Pointer();
3580     for (int i = 0; i < fNumS; i++)
3581     {
3582         ftau_NEQ[0] =  *pstress++;
3583         ftau_NEQ[1] =  *pstress++;
3584         ftau_NEQ[2] =  *pstress++;
3585 
3586         ftau_BACK[0] = *pbackstress++;
3587         ftau_BACK[1] = *pbackstress++;
3588         ftau_BACK[2] = *pbackstress++;
3589 
3590         double epse0 = 0.5*log(*ple++);
3591         double epse1 = 0.5*log(*ple++);
3592         double epse2 = 0.5*log(*ple++);
3593 
3594         double epstr0 = 0.5*log(*pltr++);
3595         double epstr1 = 0.5*log(*pltr++);
3596         double epstr2 = 0.5*log(*pltr++);
3597 
3598         double epse0N = 0.5*log(*pavec++);
3599         double epse1N = 0.5*log(*pavec++);
3600         double epse2N = 0.5*log(*pavec++);
3601 
3602         double epstr0N = 0.5*log(*pavectr++);
3603         double epstr1N = 0.5*log(*pavectr++);
3604         double epstr2N = 0.5*log(*pavectr++);
3605 
3606         double plasticwk = -(ftau_NEQ[0]*(epse0-epstr0)+ ftau_NEQ[1]*(
                epse1-epstr1) + ftau_NEQ[2]*(epse2-epstr2));
3607         plasticw += plasticwk;
3608 
3609         double backwk = ftau_BACK[0]*(epse0N-epstr0N)+ ftau_BACK[1]*(
                epse1N-epstr1N) + ftau_BACK[2]*(epse2N-epstr2N);
3610         backw += backwk;
3611     }
3612 
3613     /*add contribution from plastic work*/
3614     heat += -(plasticw-backw);
3615 }
3616 
```

```
3617 /*Reorder Eigenvalues in order of RefVec*/
3618 int ET_multi::Reorder(dArrayT& EigValue, dArrayT RefValue, dMatrixT&
     EigVec, dMatrixT& RefVec)
3619 {
3620     const double kEigenSmall = 1.0e-8;
3621     int flag = -1;
3622     int rflag = -1;
3623     int FLAG=0;
3624     bool deig0=  abs(EigValue[1]-EigValue[2])<kEigenSmall;
3625     bool deig1=  abs(EigValue[2]-EigValue[0])<kEigenSmall;
3626     bool deig2=  abs(EigValue[0]-EigValue[1])<kEigenSmall;
3627     bool deigr0= abs(RefValue[1]-RefValue[2])<kEigenSmall;
3628     bool deigr1= abs(RefValue[2]-RefValue[0])<kEigenSmall;
3629     bool deigr2= abs(RefValue[0]-RefValue[1])<kEigenSmall;
3630
3631     if (deig0 && deig1 && deig2  ) {return FLAG;}
3632     else if (deig0){
3633         flag=0;
3634     }else if (deig1){
3635         flag=1;
3636     }else if (deig2){
3637         flag=2;
3638     }
3639
3640     int order0=10; int order1=10; int order2=10;
3641
3642     double innerdot11, innerdot12, innerdot13;
3643     double innerdot21, innerdot22, innerdot23;
3644     double innerdot31, innerdot32, innerdot33;
3645
3646     innerdot11=EigVec[0]*RefVec[0]+EigVec[1]*RefVec[1]+EigVec[2]*RefVec
         [2];
3647     innerdot12=EigVec[0]*RefVec[3]+EigVec[1]*RefVec[4]+EigVec[2]*RefVec
         [5];
3648     innerdot13=EigVec[0]*RefVec[6]+EigVec[1]*RefVec[7]+EigVec[2]*RefVec
         [8];
3649     innerdot11=innerdot11*innerdot11;
3650     innerdot12=innerdot12*innerdot12;
3651     innerdot13=innerdot13*innerdot13;
3652     innerdot21=EigVec[3]*RefVec[0]+EigVec[4]*RefVec[1]+EigVec[5]*RefVec
         [2];
3653     innerdot22=EigVec[3]*RefVec[3]+EigVec[4]*RefVec[4]+EigVec[5]*RefVec
         [5];
3654     innerdot23=EigVec[3]*RefVec[6]+EigVec[4]*RefVec[7]+EigVec[5]*RefVec
         [8];
3655     innerdot21=innerdot21*innerdot21;
3656     innerdot22=innerdot22*innerdot22;
3657     innerdot23=innerdot23*innerdot23;
3658     innerdot31=EigVec[6]*RefVec[0]+EigVec[7]*RefVec[1]+EigVec[8]*RefVec
         [2];
3659     innerdot32=EigVec[6]*RefVec[3]+EigVec[7]*RefVec[4]+EigVec[8]*RefVec
         [5];
3660     innerdot33=EigVec[6]*RefVec[6]+EigVec[7]*RefVec[7]+EigVec[8]*RefVec
         [8];
3661     innerdot31=innerdot31*innerdot31;
3662     innerdot32=innerdot32*innerdot32;
```

```
3663     innerdot33=innerdot33*innerdot33;
3664
3665     if(innerdot11>innerdot12 && innerdot11>innerdot13)
3666     {
3667         order0=0;
3668     }
3669     else if (innerdot12>innerdot11 && innerdot12>innerdot13)
3670     {
3671         order0=1;
3672     }
3673     else {order0=2;}
3674
3675     if(innerdot21>innerdot22 && innerdot21>innerdot23)
3676     {
3677         order1=0;
3678     }
3679     else if (innerdot22>innerdot21 && innerdot22>innerdot23)
3680     {
3681         order1=1;
3682     }
3683     else {order1=2;}
3684
3685     if(innerdot31>innerdot32 && innerdot31>innerdot33)
3686     {
3687         order2=0;
3688     }
3689     else if (innerdot32>innerdot31 && innerdot32>innerdot33)
3690     {
3691         order2=1;
3692     }
3693     else {order2=2;}
3694     if (order0+order1+order2!=3)
3695     {
3696         cout<<"\nEle#:"<<CurrElementNumber()<<"\tIP#:"<<CurrIP()<<"\n";
3697         ExceptionT::GeneralFail("ET_multi::Reorder", "wrong ordering");
3698     }
3699
3700     if (order0!=0 || order1!=1 || order2!=2)
3701      {
3702          double temp0, temp1, temp2;
3703          temp0=EigValue[0]; temp1=EigValue[1]; temp2=EigValue[2];
3704          EigValue[order0]=temp0;
3705          EigValue[order1]=temp1;
3706          EigValue[order2]=temp2;
3707      }
3708     return FLAG;
3709 }
```

Bibliography

[1] F. Carpi, D. De Rossi, R. Kornbluh, R. E. Pelrine, and P. Sommer-Larsen, *Dielectric elastomers as electromechanical transducers: Fundamentals, materials, devices, models and applications of an emerging electroactive polymer technology*. Elsevier, 2011.

[2] Y. Bar-Cohen, *Electroactive Polymer (EAP) Actuators as Artificial Muscles*. SPIE, 2004.

[3] C. Yoon, R. Xiao, J. Park, J. Cha, T. D. Nguyen, and D. H. Gracias, "Functional stimuli responsive hydrogel devices by self-folding," *Smart Materials and Structures*, vol. 23, no. 9, p. 094008, 2014.

[4] C. Majidi, "Soft robotics: a perspectivecurrent trends and prospects for the future," *Soft Robotics*, vol. 1, no. 1, pp. 5–11, 2014.

[5] D. J. Leo, *Engineering analysis of smart material systems*. John Wiley & Sons, 2007.

[6] M. Otake, Y. Kagami, M. Inaba, and H. Inoue, "Motion design of a starfish-shaped gel robot made of electro-active polymer gel," *Robotics and Autonomous Systems*, vol. 40, no. 2-3, pp. 185–191, 2002.

[7] Z. Suo, "Theory of dielectric elastomers," *Acta Mechanica Solida Sinica*, vol. 23, no. 6, pp. 549–578, 2010.

[8] G. Kofod, P. Sommer-Larsen, R. Kornbluh, and R. Pelrine, "Actuation response of polyacrylate dielectric elastomers," *Journal of intelligent material systems and structures*, vol. 14, no. 12, pp. 787–793, 2003.

[9] M. R. Aguilar and J. San Román, *Smart polymers and their applications*. Elsevier, 2014.

[10] H. G. Schild, "Poly (n-isopropylacrylamide): experiment, theory and application," *Progress in polymer science*, vol. 17, no. 2, pp. 163–249, 1992.

[11] C. Pietsch, U. S. Schubert, and R. Hoogenboom, "Aqueous polymeric sensors based on temperature-induced polymer phase transitions and solvatochromic dyes," *Chemical Communications*, vol. 47, no. 31, pp. 8750–8765, 2011.

[12] J.-T. Zhang, S. Petersen, M. Thunga, E. Leipold, R. Weidisch, X. Liu, A. Fahr, and K. D. Jandt, "Micro-structured smart hydrogels with enhanced protein loading and release efficiency," *Acta Biomaterialia*, vol. 6, no. 4, pp. 1297–1306, 2010.

[13] A. Lendlein, M. Behl, B. Hiebl, and C. Wischke, "Shape-memory polymers as a

technology platform for biomedical applications," *Expert Review of Medical Devices*, vol. 7, no. 3, pp. 357–379, 2010.

[14] R. J. Young and P. A. Lovell, *Introduction to polymers*. CRC press, 2011.

[15] H. Van Melick, L. Govaert, B. Raas, W. Nauta, and H. Meijer, "Kinetics of ageing and re-embrittlement of mechanically rejuvenated polystyrene," *Polymer*, vol. 44, no. 4, pp. 1171–1179, 2003.

[16] O. Hasan and M. C. Boyce, "Energy storage during inelastic deformation of glassy polymers," *Polymer*, vol. 34, no. 24, pp. 5085–5092, 1993.

[17] D. Senden, J. Van Dommelen, and L. Govaert, "Strain hardening and its relation to bauschinger effects in oriented polymers," *Journal of Polymer Science Part B: Polymer Physics*, vol. 48, no. 13, pp. 1483–1494, 2010.

[18] D. J. Senden, S. Krop, J. van Dommelen, and L. Govaert, "Rate-and temperature-dependent strain hardening of polycarbonate," *Journal of Polymer Science Part B: Polymer Physics*, vol. 50, no. 24, pp. 1680–1693, 2012.

[19] E. M. Arruda, H. Quintus-Bosz, and M. C. Boyce, "Effects of initial anisotropy on the finite strain deformation behavior of glassy polymers," in *Anisotropy and Localization of Plastic Deformation*. Springer, 1991, pp. 489–492.

[20] K. Asaka and H. Okuzaki, *Soft Actuators: Materials, Modeling, Applications, and Future Perspectives*. Springer, 2014.

BIBLIOGRAPHY

[21] L. S. Hirst, *Fundamentals of soft matter science*. CRC Press, 2012.

[22] P. de Gennes, "soft matter: the birth and growth of concepts," *Twentieth Century Physics. London*, vol. 1593, 1995.

[23] A. Katchalsky and M. Zwick, "Mechanochemistry and ion exchange," *Journal of Polymer Science*, vol. 16, no. 82, pp. 221–234, 1955.

[24] Y. Osada, "Conversion of chemical into mechanical energy by synthetic polymers (chemomechanical systems)," in *Polymer Physics*. Springer, 1987, pp. 1–46.

[25] R. Baughman, L. Shacklette, R. Elsenbaumer, E. Plichta, and C. Becht, "Conducting polymer electromechanical actuators," in *Conjugated Polymeric Materials: Opportunities in Electronics, Optoelectronics, and Molecular Electronics*. Springer, 1990, pp. 559–582.

[26] ——, "Micro electromechanical actuators based on conducting polymers," in *Molecular Electronics*. Springer, 1991, pp. 267–289.

[27] D. DeRossi, K. Kajiwara, Y. Osada, and A. Yamauchi, "Polymer gels," in *Fundamentals and Biomedical Applications*. Springer, 1991.

[28] T. Otero and J. Rodríguez, "Electrochemomechanical and electrochemopositioning devices: artificial muscles," in *Intrinsically conducting polymers: an emerging technology*. Springer, 1993, pp. 179–190.

[29] Y. Osada and J.-P. Gong, "Soft and wet materials: polymer gels," *Advanced Materials*, vol. 10, no. 11, pp. 827–837, 1998.

[30] E. Smela, "Conjugated polymer actuators for biomedical applications," *Advanced materials*, vol. 15, no. 6, pp. 481–494, 2003.

[31] J. D. Madden, N. A. Vandesteeg, P. A. Anquetil, P. G. Madden, A. Takshi, R. Z. Pytel, S. R. Lafontaine, P. A. Wieringa, and I. W. Hunter, "Artificial muscle technology: physical principles and naval prospects," *IEEE Journal of oceanic engineering*, vol. 29, no. 3, pp. 706–728, 2004.

[32] P. Brochu and Q. Pei, "Advances in dielectric elastomers for actuators and artificial muscles," *Macromolecular rapid communications*, vol. 31, no. 1, pp. 10–36, 2010.

[33] H. Okuzaki, T. Kuwabara, K. Funasaka, and T. Saido, "Humidity-sensitive polypyrrole films for electro-active polymer actuators," *Advanced Functional Materials*, vol. 23, no. 36, pp. 4400–4407, 2013.

[34] G. Chapman, "The hydrostatic skeleton in the invertebrates," *Biological reviews*, vol. 33, no. 3, pp. 338–371, 1958.

[35] ——, "Versatility of hydraulic systems," *Journal of experimental Zoology*, vol. 194, no. 1, pp. 249–269, 1975.

[36] R. B. Clark, *Dynamics in metazoan evolution: the origin of the coelom and segments*. Oxford University Press, 1964.

[37] W. F. Gutmann, "Relationships between invertebrate phyla based on functional-mechanical analysis of the hydrostatic skeleton," *American zoologist*, vol. 21, no. 1, pp. 63–81, 1981.

[38] W. M. Kier and K. K. Smith, "Tongues, tentacles and trunks: the biomechanics of movement in muscular-hydrostats," *Zoological journal of the Linnean Society*, vol. 83, no. 4, pp. 307–324, 1985.

[39] W. M. Kier, "The functional morphology of the tentacle musculature of nautilus pompilius," in *Nautilus*. Springer, 2010, pp. 257–269.

[40] ——, "The fin musculature of cuttlefish and squid (mollusca, cephalopoda): morphology and mechanics," *Journal of Zoology*, vol. 217, no. 1, pp. 23–38, 1989.

[41] W. M. Kier and A. M. Smith, "The morphology and mechanics of octopus suckers," *The Biological Bulletin*, vol. 178, no. 2, pp. 126–136, 1990.

[42] A. Herrel, J. J. Meyers, P. Aerts, and K. C. Nishikawa, "Functional implications of supercontracting muscle in the chameleon tongue retractors," *Journal of Experimental Biology*, vol. 204, no. 21, pp. 3621–3627, 2001.

[43] J. J. Meyers, J. C. O'Reilly, J. A. Monroy, and K. C. Nishikawa, "Mechanism of tongue protraction in microhylid frogs," *Journal of experimental biology*, vol. 207, no. 1, pp. 21–31, 2004.

[44] V. J. Napadow, R. D. Kamm, and R. J. Gilbert, "A biomechanical model of sagittal

tongue bending," *Journal of biomechanical engineering*, vol. 124, no. 5, pp. 547–556, 2002.

[45] K. C. Nishikawa, W. M. Kier, and K. K. Smith, "Morphology and mechanics of tongue movement in the african pig-nosed frog hemisus marmoratum: a muscular hydrostatic model," *Journal of Experimental Biology*, vol. 202, no. 7, pp. 771–780, 1999.

[46] D. Trivedi, C. D. Rahn, W. M. Kier, and I. D. Walker, "Soft robotics: Biological inspiration, state of the art, and future research," *Applied bionics and biomechanics*, vol. 5, no. 3, pp. 99–117, 2008.

[47] R. Pelrine, R. D. Kornbluh, Q. Pei, S. Stanford, S. Oh, J. Eckerle, R. J. Full, M. A. Rosenthal, and K. Meijer, "Dielectric elastomer artificial muscle actuators: toward biomimetic motion," in *SPIE's 9th Annual International Symposium on Smart Structures and Materials*. International Society for Optics and Photonics, 2002, pp. 126–137.

[48] Q. Pei, M. A. Rosenthal, R. Pelrine, S. Stanford, and R. D. Kornbluh, "Multifunctional electroelastomer roll actuators and their application for biomimetic walking robots," in *Smart Structures and Materials 2003: Electroactive Polymer Actuators and Devices (EAPAD)*, vol. 5051. International Society for Optics and Photonics, 2003, pp. 281–291.

[49] Q. Pei, R. Pelrine, S. Stanford, R. D. Kornbluh, M. S. Rosenthal, K. Meijer, and R. J.

Full, "Multifunctional electroelastomer rolls and their application for biomimetic walking robots," in *Smart Structures and Materials 2002: Industrial and Commercial Applications of Smart Structures Technologies*, vol. 4698. International Society for Optics and Photonics, 2002, pp. 246–254.

[50] Q. Pei, M. Rosenthal, S. Stanford, H. Prahlad, and R. Pelrine, "Multiple-degrees-of-freedom electroelastomer roll actuators," *Smart materials and structures*, vol. 13, no. 5, p. N86, 2004.

[51] M. A. C. Stuart, W. T. Huck, J. Genzer, M. Müller, C. Ober, M. Stamm, G. B. Sukhorukov, I. Szleifer, V. V. Tsukruk, M. Urban *et al.*, "Emerging applications of stimuli-responsive polymer materials," *Nature materials*, vol. 9, no. 2, p. 101, 2010.

[52] C. De las Heras Alarcón, S. Pennadam, and C. Alexander, "Stimuli responsive polymers for biomedical applications," *Chemical Society Reviews*, vol. 34, no. 3, pp. 276–285, 2005.

[53] D. Schmaljohann, "Thermo-and ph-responsive polymers in drug delivery," *Advanced drug delivery reviews*, vol. 58, no. 15, pp. 1655–1670, 2006.

[54] M. A. Ward and T. K. Georgiou, "Thermoresponsive polymers for biomedical applications," *Polymers*, vol. 3, no. 3, pp. 1215–1242, 2011.

[55] M. Heskins and J. E. Guillet, "Solution properties of poly (n-isopropylacrylamide)," *Journal of Macromolecular ScienceChemistry*, vol. 2, no. 8, pp. 1441–1455, 1968.

[56] Z. Hu, X. Zhang, and Y. Li, "Synthesis and application of modulated polymer gels," *Science*, vol. 269, no. 5223, pp. 525–527, 1995.

[57] S. Maeda, Y. Hara, T. Sakai, R. Yoshida, and S. Hashimoto, "Self-walking gel," *Advanced Materials*, vol. 19, no. 21, pp. 3480–3484, 2007.

[58] X. Zhang, C. L. Pint, M. H. Lee, B. E. Schubert, A. Jamshidi, K. Takei, H. Ko, A. Gillies, R. Bardhan, J. J. Urban *et al.*, "Optically-and thermally-responsive programmable materials based on carbon nanotube-hydrogel polymer composites," *Nano letters*, vol. 11, no. 8, pp. 3239–3244, 2011.

[59] S. Fusco, M. S. Sakar, S. Kennedy, C. Peters, R. Bottani, F. Starsich, A. Mao, G. A. Sotiriou, S. Pané, S. E. Pratsinis *et al.*, "An integrated microrobotic platform for on-demand, targeted therapeutic interventions," *Advanced Materials*, vol. 26, no. 6, pp. 952–957, 2014.

[60] N. Bassik, B. T. Abebe, K. E. Laflin, and D. H. Gracias, "Photolithographically patterned smart hydrogel based bilayer actuators," *Polymer*, vol. 51, no. 26, pp. 6093–6098, 2010.

[61] N. Bassik, B. T. Abebe, and D. H. Gracias, "Solvent driven motion of lithographically fabricated gels," *Langmuir*, vol. 24, no. 21, pp. 12 158–12 163, 2008.

[62] S. Ghosh and T. Cai, "Controlled actuation of alternating magnetic field-sensitive

tunable hydrogels," *Journal of Physics D: Applied Physics*, vol. 43, no. 41, p. 415504, 2010.

[63] R. Yoshida, K. Omata, K. Yamaura, M. Ebata, M. Tanaka, and M. Takai, "Maskless microfabrication of thermosensitive gels using a microscope and application to a controlled release microchip," *Lab on a Chip*, vol. 6, no. 10, pp. 1384–1386, 2006.

[64] L.-Y. Chu, J.-W. Kim, R. K. Shah, and D. A. Weitz, "Monodisperse thermoresponsive microgels with tunable volume-phase transition kinetics," *Advanced Functional Materials*, vol. 17, no. 17, pp. 3499–3504, 2007.

[65] Y. Zhang and A. L. Yarin, "Stimuli-responsive copolymers of n-isopropyl acrylamide with enhanced longevity in water for micro-and nanofluidics, drug delivery and non-woven applications," *Journal of Materials Chemistry*, vol. 19, no. 27, pp. 4732–4739, 2009.

[66] A. Pelah, A. Bharde, and T. M. Jovin, "Protein manipulation by stimuli-responsive polymers encapsulated in erythrocyte ghosts," *Soft Matter*, vol. 5, no. 5, pp. 1006–1010, 2009.

[67] S. Zakharchenko, N. Puretskiy, G. Stoychev, M. Stamm, and L. Ionov, "Temperature controlled encapsulation and release using partially biodegradable thermo-magneto-sensitive self-rolling tubes," *Soft Matter*, vol. 6, no. 12, pp. 2633–2636, 2010.

[68] G. Stoychev, N. Puretskiy, and L. Ionov, "Self-folding all-polymer thermoresponsive microcapsules," *Soft Matter*, vol. 7, no. 7, pp. 3277–3279, 2011.

[69] K. L. Fujimoto, Z. Ma, D. M. Nelson, R. Hashizume, J. Guan, K. Tobita, and W. R. Wagner, "Synthesis, characterization and therapeutic efficacy of a biodegradable, thermoresponsive hydrogel designed for application in chronic infarcted myocardium," *Biomaterials*, vol. 30, no. 26, pp. 4357–4368, 2009.

[70] S. Zhang, Y. Feng, L. Zhang, J. Sun, X. Xu, and Y. Xu, "Novel interpenetrating networks with shape-memory properties," *Journal of Polymer Science Part A: Polymer Chemistry*, vol. 45, no. 5, pp. 768–775, 2007.

[71] M. Behl, U. Ridder, Y. Feng, S. Kelch, and A. Lendlein, "Shape-memory capability of binary multiblock copolymer blends with hard and switching domains provided by different components," *Soft Matter*, vol. 5, no. 3, pp. 676–684, 2009.

[72] A. Lendlein and S. Kelch, "Degradable, multifunctional polymeric biomaterials with shape-memory," in *Materials Science Forum*, vol. 492. Trans Tech Publ, 2005, pp. 219–224.

[73] M. Everhart and J. Stahl, "High strain fiber reinforced reusable shape memory polymer mandrels," in *Int. SAMPE Symp. Exhib*, vol. 50, 2005, pp. 955–965.

[74] T. Xie and X. Xiao, "Self-peeling reversible dry adhesive system," *Chemistry of Materials*, vol. 20, no. 9, pp. 2866–2868, 2008.

[75] H. Hussein and D. Harrison, *Investigation into the use of engineering polymers as acutators to produce automatic disassemblyof electronic products*. Wiley-VCH, Weinheim, 2004.

[76] A. Q. Tool, "Relaxation of stresses in annealing glass," *J. Res. Natl. Bur. Stand.(US)*, vol. 34, no. 2, pp. 199–211, 1945.

[77] L. Struik, "Physical aging in amorphous glassy polymers," *Annals of the New York Academy of Sciences*, vol. 279, no. 1, pp. 78–85, 1976.

[78] J. M. Hutchinson, "Physical aging of polymers," *Progress in Polymer Science*, vol. 20, no. 4, pp. 703–760, 1995.

[79] I. M. Hodge, "Physical aging in polymer glasses," *Science*, vol. 267, no. 5206, pp. 1945–1947, 1995.

[80] H. E. Meijer and L. E. Govaert, "Mechanical performance of polymer systems: The relation between structure and properties," *Progress in polymer science*, vol. 30, no. 8-9, pp. 915–938, 2005.

[81] O. Hasan and M. Boyce, "A constitutive model for the nonlinear viscoelastic viscoplastic behavior of glassy polymers," *Polymer Engineering & Science*, vol. 35, no. 4, pp. 331–344, 1995.

[82] R. Xiao and T. D. Nguyen, "An effective temperature theory for the nonequilibrium

behavior of amorphous polymers," *Journal of the Mechanics and Physics of Solids*, vol. 82, pp. 62–81, 2015.

[83] J. Guo, R. Xiao, H. S. Park, and T. D. Nguyen, "The temperature-dependent viscoelastic behavior of dielectric elastomers," *Journal of Applied Mechanics*, vol. 82, no. 9, p. 091009, 2015.

[84] I. A. Anderson, T. A. Gisby, T. G. McKay, B. M. O'Brien, and E. P. Calius, "Multi-functional dielectric elastomer artificial muscles for soft and smart machines," *Journal of Applied Physics*, vol. 112, no. 4, p. 041101, 2012.

[85] R. Pelrine, R. Kornbluh, Q. Pei, and J. Joseph, "High-speed electrically actuated elastomers with strain greater than 100%," *Science*, vol. 287, no. 5454, pp. 836–839, 2000.

[86] A. O'Halloran, F. O'Malley, and P. McHugh, "A review on dielectric elastomer actuators, technology, applications, and challenges," *Journal of Applied Physics*, vol. 104, no. 7, p. 071101, 2008.

[87] E. Biddiss and T. Chau, "Dielectric elastomers as actuators for upper limb prosthetics: Challenges and opportunities," *Medical engineering & physics*, vol. 30, no. 4, pp. 403–418, 2008.

[88] S. M. Ha, W. Yuan, Q. Pei, R. Pelrine, and S. Stanford, "Interpenetrating polymer

networks for high-performance electroelastomer artificial muscles," *Advanced Materials*, vol. 18, no. 7, pp. 887–891, 2006.

[89] R. Pelrine, R. D. Kornbluh, J. Eckerle, P. Jeuck, S. Oh, Q. Pei, and S. Stanford, "Dielectric elastomers: generator mode fundamentals and applications," in *SPIE's 8th Annual International Symposium on Smart Structures and Materials*. International Society for Optics and Photonics, 2001, pp. 148–156.

[90] S. J. A. Koh, X. Zhao, and Z. Suo, "Maximal energy that can be converted by a dielectric elastomer generator," *Applied Physics Letters*, vol. 94, no. 26, p. 262902, 2009.

[91] M. Beck, R. Fiolka, and A. Stemmer, "Variable phase retarder made of a dielectric elastomer actuator," *Optics letters*, vol. 34, no. 6, pp. 803–805, 2009.

[92] G. Kofod, D. N. Mc Carthy, J. Krissler, G. Lang, and G. Jordan, "Electroelastic optical fiber positioning with submicrometer accuracy: Model and experiment," *Applied Physics Letters*, vol. 94, no. 20, p. 202901, 2009.

[93] C. Keplinger, M. Kaltenbrunner, N. Arnold, and S. Bauer, "Röntgen's electrode-free elastomer actuators without electromechanical pull-in instability," *Proceedings of the National Academy of Sciences*, vol. 107, no. 10, pp. 4505–4510, 2010.

[94] Y. Bar-Cohen, "Electroactive polymers for refreshable braille displays," *SPIE Newsroom*, 2009.

[95] F. Carpi, G. Frediani, and D. De Rossi, "Electroactive elastomeric haptic displays of organ motility and tissue compliance for medical training and surgical force feedback," *Biomedical Engineering, IEEE Transactions on*, vol. 56, no. 9, pp. 2327–2330, 2009.

[96] ——, "Hydrostatically coupled dielectric elastomer actuators for tactile displays and cutaneous stimulators," in *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*. International Society for Optics and Photonics, 2010, pp. 76 420E–76 420E.

[97] J.-S. Plante and S. Dubowsky, "Large-scale failure modes of dielectric elastomer actuators," *International Journal of Solids and Structures*, vol. 43, no. 25, pp. 7727–7751, 2006.

[98] S. J. A. Koh, T. Li, J. Zhou, X. Zhao, W. Hong, J. Zhu, and Z. Suo, "Mechanisms of large actuation strain in dielectric elastomers," *Journal of Polymer Science Part B: Polymer Physics*, vol. 49, no. 7, pp. 504–515, 2011.

[99] G. Kofod, "The static actuation of dielectric elastomer actuators: how does pre-stretch improve actuation?" *Journal of Physics D: Applied Physics*, vol. 41, no. 21, p. 215405, 2008.

[100] Z. Suo, X. Zhao, and W. H. Greene, "A nonlinear field theory of deformable dielectrics," *Journal of the Mechanics and Physics of Solids*, vol. 56, no. 2, pp. 467–486, 2008.

[101] T. Lu, J. Huang, C. Jordi, G. Kovacs, R. Huang, D. R. Clarke, and Z. Suo, "Dielectric elastomer actuators under equal-biaxial forces, uniaxial forces, and uniaxial constraint of stiff fibers," *Soft Matter*, vol. 8, no. 22, pp. 6167–6173, 2012.

[102] N. Goulbourne, E. Mockensturm, and M. Frecker, "A nonlinear model for dielectric elastomer membranes," *Journal of Applied Mechanics*, vol. 72, no. 6, pp. 899–906, 2005.

[103] R. M. McMeeking and C. M. Landis, "Electrostatic forces and stored energy for deformable dielectric materials," *Journal of Applied Mechanics*, vol. 72, no. 4, pp. 581–590, 2005.

[104] Y.-Y. Jhong, C.-M. Huang, C.-C. Hsieh, and C.-C. Fu, "Improvement of viscoelastic effects of dielectric elastomer actuator and its application for valve devices," in *The 14th International Symposium on: Smart Structures and Materials & Nondestructive Evaluation and Health Monitoring*. International Society for Optics and Photonics, 2007, pp. 65 241Y–65 241Y.

[105] R. Palakodeti and M. Kessler, "Influence of frequency and prestrain on the mechanical efficiency of dielectric electroactive polymer actuators," *Materials Letters*, vol. 60, no. 29, pp. 3437–3440, 2006.

[106] C. Keplinger, M. Kaltenbrunner, N. Arnold, and S. Bauer, "Capacitive extensometry for transient strain analysis of dielectric elastomer actuators," *Applied Physics Letters*, vol. 92, no. 19, p. 192903, 2008.

[107] J. Zhang and H. Chen, "Electromechanical performance of a viscoelastic dielectric elastomer balloon," *International Journal of Smart and Nano Materials*, vol. 5, no. 2, pp. 76–85, 2014.

[108] X. Zhao, S. J. A. Koh, and Z. Suo, "Nonequilibrium thermodynamics of dielectric elastomers," *International journal of applied mechanics*, vol. 3, no. 02, pp. 203–217, 2011.

[109] H. S. Park and T. D. Nguyen, "Viscoelastic effects on electromechanical instabilities in dielectric elastomers," *Soft Matter*, vol. 9, no. 4, pp. 1031–1042, 2013.

[110] J. Wang, T. D. Nguyen, and H. S. Park, "Electrostatically driven creep in viscoelastic dielectric elastomers," *Journal of Applied Mechanics*, vol. 81, no. 5, p. 051006, 2014.

[111] P. Lochmatter, G. Kovacs, and M. Wissler, "Characterization of dielectric elastomer actuators based on a visco-hyperelastic film model," *Smart Materials and Structures*, vol. 16, no. 2, p. 477, 2007.

[112] E. Yang, M. Frecker, and E. Mockensturm, "Viscoelastic model of dielectric elastomer membranes," in *Smart Structures and Materials*. International Society for Optics and Photonics, 2005, pp. 82–93.

[113] M. Wissler and E. Mazza, "Mechanical behavior of an acrylic elastomer used in

dielectric elastomer actuators," *Sensors and Actuators A: Physical*, vol. 134, no. 2, pp. 494–504, 2007.

[114] M. Kollosche, G. Kofod, Z. Suo, and J. Zhu, "Temporal evolution and instability in a viscoelastic dielectric elastomer," *Journal of the Mechanics and Physics of Solids*, vol. 76, pp. 47–64, 2015.

[115] M. Wissler and E. Mazza, "Modeling and simulation of dielectric elastomer actuators," *Smart Materials and structures*, vol. 14, no. 6, p. 1396, 2005.

[116] S. Michel, X. Q. Zhang, M. Wissler, C. Löwe, and G. Kovacs, "A comparison between silicone and acrylic elastomers as dielectric materials in electroactive polymer actuators," *Polymer international*, vol. 59, no. 3, pp. 391–399, 2010.

[117] T. He, X. Zhao, and Z. Suo, "Dielectric elastomer membranes undergoing inhomogeneous deformation," *Journal of Applied Physics*, vol. 106, no. 8, p. 083522, 2009.

[118] J. Leng, L. Liu, Y. Liu, K. Yu, and S. Sun, "Electromechanical stability of dielectric elastomer," *Applied Physics Letters*, vol. 94, no. 21, p. 211901, 2009.

[119] A. Gent, "A new constitutive relation for rubber," *Rubber chemistry and technology*, vol. 69, no. 1, pp. 59–61, 1996.

[120] S. Reese and S. Govindjee, "A theory of finite viscoelasticity and numerical aspects," *Int. J. Solids Struct.*, vol. 35, pp. 3455–82, 1998.

[121] F. Schwarzl and A. Staverman, "Higher approximation methods for the relaxation spectrum from static and dynamic measurements of visco-elastic materials," *Applied Scientific Research, Section A*, vol. 4, no. 2, pp. 127–141, 1953.

[122] R. Xiao, J. Guo, and T. D. Nguyen, "Modeling the multiple shape memory effect and temperature memory effect in amorphous polymers," *RSC Advances*, vol. 5, no. 1, pp. 416–423, 2015.

[123] J. D. Ferry, *Viscoelastic Properties of Polymers*. New York, NY: John Wiley and Sons, 1980.

[124] T. Nguyen, C. M. Yakacki, P. D. Brahmbhatt, M. L. Chambers *et al.*, "Modeling the relaxation mechanisms of amorphous shape memory polymers," *Advanced Materials*, vol. 22, no. 31, pp. 3411–3423, 2010.

[125] P. Haupt, A. Lion, and E. Backhaus, "On the dynamic behaviour of polymers under finite strains: constitutive modelling and identification of parameters," *Int. J. Solids Struct.*, vol. 37, no. 26, pp. 3633–3646, 2000.

[126] C. C. Foo, S. Cai, S. J. A. Koh, S. Bauer, and Z. Suo, "Model of dissipative dielectric elastomers," *Journal of Applied Physics*, vol. 111, no. 3, p. 034102, 2012.

[127] J. Guo, T. Shroff, C. Yoon, J. Liu, J. C. Breger, D. H. Gracias, and T. D. Nguyen, "Bidirectional and biaxial curving of thermoresponsive bilayer plates with soft and stiff segments," *Extreme Mechanics Letters*, vol. 16, pp. 6–12, 2017.

BIBLIOGRAPHY

[128] D. Fengel and G. Wegener, *Wood: chemistry, ultrastructure, reactions*. Walter de Gruyter, 1983.

[129] M. McNeil, A. G. Darvill, S. C. Fry, and P. Albersheim, "Structure and function of the primary cell walls of plants," *Annual review of biochemistry*, vol. 53, no. 1, pp. 625–663, 1984.

[130] I. Burgert and P. Fratzl, "Actuation systems in plants as prototypes for bioinspired devices," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1893, pp. 1541–1557, 2009.

[131] C. Dawson, J. F. Vincent, and A.-M. Rocca, "How pine cones open," *Nature*, vol. 390, no. 6661, pp. 668–668, 1997.

[132] C. Ingold, "Peristome teeth and spore discharge in mosses," in *Transactions of the Botanical Society of Edinburgh*, vol. 38, no. 1-4. Taylor & Francis, 1959, pp. 76–88.

[133] W. Haupt, *Bewegungsphysiologie der Pflanzen*. Georg Thieme, 1977.

[134] Z. Qin, N. M. Pugno, and M. J. Buehler, "Mechanics of fragmentation of crocodile skin and other thin films," *Scientific reports*, vol. 4, 2014.

[135] C. Chang, P. Wu, R. E. Baker, P. K. Maini, L. Alibardi, and C.-M. Chuong, "Reptile scale paradigm: Evo-devo, pattern formation and regeneration," *The International journal of developmental biology*, vol. 53, no. 5-6, p. 813, 2009.

BIBLIOGRAPHY

[136] M. A. Meyers, Y. Lin, E. Olevsky, and P.-Y. Chen, "Battle in the amazon: arapaima versus piranha," *Advanced Engineering Materials*, vol. 14, no. 5, pp. B279–B288, 2012.

[137] C. L. Randall, E. Gultepe, and D. H. Gracias, "Self-folding devices and materials for biomedical applications," *Trends In Biotechnology*, vol. 30, no. 3, pp. 138–146, 2012.

[138] N. B. Bowden, M. Weck, I. S. Choi, and G. M. Whitesides, "Molecule-mimetic chemistry and mesoscale self-assembly," *Accounts Of Chemical Research*, vol. 34, no. 3, pp. 231–238, 2001.

[139] S. Alben and M. P. Brenner, "Self-assembly of flat sheets into closed surfaces," *Physical Review E*, vol. 75, no. 5, p. 056113, 2007.

[140] R. Fernandes and D. H. Gracias, "Self-folding polymeric containers for encapsulation and delivery of drugs," *Advanced Drug Delivery Reviews*, vol. 64, no. 14, pp. 1579–1589, 2012.

[141] D. H. Gracias, "Stimuli responsive self-folding using thin polymer films," *Current Opinion in Chemical Engineering*, vol. 2, no. 1, pp. 112–119, 2013.

[142] T. G. Leong, B. R. Benson, E. K. Call, and D. H. Gracias, "Thin film stress driven self-folding of microstructured containers," *Small*, vol. 4, no. 10, pp. 1605–1609, 2008.

[143] A. M. Abdullah, K. Nan, J. A. Rogers, and K. J. Hsia, "Mismatch strain programmed shape transformation of curved bilayer-flexible support assembly," *Extreme Mechanics Letters*, vol. 7, pp. 34–41, 2016.

[144] A. M. Abdullah, P. V. Braun, and K. J. Hsia, "Programmable shape transformation of elastic spherical domes," *Soft matter*, vol. 12, no. 29, pp. 6184–6195, 2016.

[145] M. Jamal, A. M. Zarafshar, and D. H. Gracias, "Differentially photo-crosslinked polymers enable self-assembling microfluidics," *Nature Communications*, vol. 2, p. 527, 2011.

[146] J. Kim, J. A. Hanna, M. Byun, C. D. Santangelo, and R. C. Hayward, "Designing responsive buckled surfaces by halftone gel lithography," *Science*, vol. 335, no. 6073, pp. 1201–1205, 2012.

[147] Y. Klein, E. Efrati, and E. Sharon, "Shaping of elastic sheets by prescription of non-euclidean metrics," *Science*, vol. 315, no. 5815, pp. 1116–1120, 2007.

[148] Z. L. Wu, M. Moshe, J. Greener, H. Therien-Aubin, Z. Nie, E. Sharon, and E. Kumacheva, "Three-dimensional shape transformations of hydrogel sheets induced by small-scale modulation of internal stresses," *Nature Communications*, vol. 4, p. 1586, 2013.

[149] L. Ionov, "Soft microorigami: self-folding polymer films," *Soft Matter*, vol. 7, no. 15, pp. 6786–6791, 2011.

[150] ——, "Hydrogel-based actuators: possibilities and limitations," *Materials Today*, vol. 17, no. 10, pp. 494–503, 2014.

[151] ——, "Biomimetic hydrogel-based actuating systems," *Advanced Functional Materials*, vol. 23, no. 36, pp. 4555–4570, 2013.

[152] M. J. Motala, D. Perlitz, C. M. Daly, P. Yuan, R. G. Nuzzo, and K. J. Hsia, "Programming matter through strain," *Extreme Mechanics Letters*, vol. 3, pp. 8–16, 2015.

[153] R. M. Erb, J. S. Sander, R. Grisch, and A. R. Studart, "Self-shaping composites with programmable bioinspired microstructures," *Nature Communications*, vol. 4, p. 1712, 2013.

[154] K. Malachowski, J. Breger, H. R. Kwag, M. O. Wang, J. P. Fisher, F. M. Selaru, and D. H. Gracias, "Stimuli-responsive theragrippers for chemomechanical controlled release," *Angewandte Chemie*, vol. 126, no. 31, pp. 8183–8187, 2014.

[155] J. C. Breger, C. Yoon, R. Xiao, H. R. Kwag, M. O. Wang, J. P. Fisher, T. D. Nguyen, and D. H. Gracias, "Self-folding thermo-magnetically responsive soft microgrippers," *Acs Applied Materials & Interfaces*, vol. 7, no. 5, pp. 3398–3405, 2015.

[156] S. Turcaud, L. Guiducci, P. Fratzl, Y. J. Bréchet, and J. W. Dunlop, "An excursion into the design space of biomimetic architectured biphasic actuators," *International Journal of Materials Research*, vol. 102, no. 6, pp. 607–612, 2011.

[157] T. Shroff, "Microfabricated dynamic shape-change hydrogel systems," Master's thesis, The Johns Hopkins University, Baltimore, Maryland, 2016.

[158] N. Bhattarai, J. Gunn, and M. Zhang, "Chitosan-based hydrogels for controlled, localized drug delivery," *Advanced Drug Delivery Reviews*, vol. 62, no. 1, pp. 83–99, 2010.

[159] S. Katayama, Y. Hirokawa, and T. Tanaka, "Reentrant phase transition in acrylamide-derivative copolymer gels," *Macromolecules*, vol. 17, no. 12, pp. 2641–2643, 1984.

[160] S. Wongsuwarn, D. Vigolo, R. Cerbino, A. M. Howe, A. Vailati, R. Piazza, and P. Cicuta, "Giant thermophoresis of poly (n-isopropylacrylamide) microgel particles," *Soft Matter*, vol. 8, no. 21, pp. 5857–5863, 2012.

[161] E. S. Matsuo and T. Tanaka, "Patterns in shrinking gels," *Nature*, vol. 358, p. 483, 1992.

[162] H. Li, X. Guo, R. G. Nuzzo, and K. J. Hsia, "Capillary induced self-assembly of thin foils into 3d structures," *Journal of the Mechanics and Physics of Solids*, vol. 58, no. 12, pp. 2033–2042, 2010.

[163] X. Guo, H. Li, B. Y. Ahn, E. B. Duoss, K. J. Hsia, J. A. Lewis, and R. G. Nuzzo, "Two-and three-dimensional folding of thin film single-crystalline silicon for photo-

voltaic power applications," *Proceedings of the National Academy of Sciences*, vol. 106, no. 48, pp. 20 149–20 154, 2009.

[164] S. Alben, B. Balakrisnan, and E. Smela, "Edge effects determine the direction of bilayer bending," *Nano Letters*, vol. 11, no. 6, pp. 2280–2285, 2011.

[165] G. Stoychev, S. Zakharchenko, S. Turcaud, J. W. Dunlop, and L. Ionov, "Shape-programmed folding of stimuli-responsive polymer bilayers," *ACS Nano*, vol. 6, no. 5, pp. 3925–3934, 2012.

[166] M. W. Hyer, "Calculations of the room-temperature shapes of unsymmetric laminat-estwo," *Journal of Composite Materials*, vol. 15, no. 4, pp. 296–310, 1981.

[167] M. M. Shokrieh and S. M. Kamali, "Theoretical and experimental studies on residual stresses in laminated polymer composites," *Journal of Composite Materials*, vol. 39, no. 24, pp. 2213–2225, 2005.

[168] I. S. Chun, A. Challa, B. Derickson, K. J. Hsia, and X. Li, "Geometry effect on the strain-induced self-rolling of semiconductor membranes," *Nano Letters*, vol. 10, no. 10, pp. 3927–3932, 2010.

[169] K.-U. Jeong, J.-H. Jang, D.-Y. Kim, C. Nah, J. H. Lee, M.-H. Lee, H.-J. Sun, C.-L. Wang, S. Z. Cheng, and E. L. Thomas, "Three-dimensional actuators transformed from the programmed two-dimensional structures via bending, twisting and folding mechanisms," *Journal of Materials Chemistry*, vol. 21, no. 19, pp. 6824–6830, 2011.

[170] G. Stoychev, S. Turcaud, J. W. Dunlop, and L. Ionov, "Hierarchical multi-step folding of polymer bilayers," *Advanced Functional Materials*, vol. 23, no. 18, pp. 2295–2300, 2013.

[171] R. Xiao, "Modeling mismatch strain induced self-folding of bilayer gel structures," *International Journal of Applied Mechanics*, vol. 8, no. 07, p. 1640004, 2016.

[172] P. J. Flory and J. Rehner Jr, "Statistical mechanics of cross-linked polymer networks i. rubberlike elasticity," *The Journal of Chemical Physics*, vol. 11, no. 11, pp. 512–520, 1943.

[173] S. A. Chester and L. Anand, "A thermo-mechanically coupled theory for fluid permeation in elastomeric materials: application to thermally responsive gels," *Journal of the Mechanics and Physics of Solids*, vol. 59, no. 10, pp. 1978–2006, 2011.

[174] H. Eyring, "Viscosity, plasticity, and diffusion as examples of absolute reaction rates," *The Journal of chemical physics*, vol. 4, no. 4, pp. 283–291, 1936.

[175] C. G'sell, "Plastic deformation of glassy polymers: constitutive equations and macromolecular mechanisms," in *Strength of Metals and Alloys (ICSMA 7)*. Elsevier, 1986, pp. 1943–1982.

[176] J.-C. Bauwens, "A new approach to describe the tensile stress-strain curve of a glassy polymer," *Journal of Materials Science*, vol. 13, no. 7, pp. 1443–1448, 1978.

[177] L. Govaert, P. Timmermans, and W. Brekelmans, "The influence of intrinsic strain

softening on strain localization in polycarbonate: modeling and experimental validation," *Journal of Engineering Materials and Technology*, vol. 122, no. 2, pp. 177–185, 2000.

[178] L. Govaert, H. Van Melick, and H. Meijer, "Temporary toughening of polystyrene through mechanical pre-conditioning," *Polymer*, vol. 42, no. 3, pp. 1271–1274, 2001.

[179] E. Klompen, T. Engels, L. Govaert, and H. Meijer, "Modeling of the postyield response of glassy polymers: influence of thermomechanical history," *Macromolecules*, vol. 38, no. 16, pp. 6997–7008, 2005.

[180] G. Adam and J. H. Gibbs, "On the temperature dependence of cooperative relaxation properties in glass-forming liquids," *The journal of chemical physics*, vol. 43, no. 1, pp. 139–146, 1965.

[181] H. Van Melick, L. Govaert, and H. Meijer, "Localisation phenomena in glassy polymers: influence of thermal and mechanical history," *Polymer*, vol. 44, no. 12, pp. 3579–3591, 2003.

[182] H. E. Meijer, L. E. Govaert, R. J. Smit *et al.*, "A multi-level finite element method for modeling rubber-toughened amorphous polymers," in *ACS Symposium Series*, vol. 759. Citeseer, 2000, pp. 50–70.

[183] E. J. Kramer, "Open questions in the physics of deformation of polymer glasses,"

Journal of Polymer Science Part B: Polymer Physics, vol. 43, no. 23, pp. 3369–3371, 2005.

[184] R. Haward and G. . Thackray, "The use of a mathematical model to describe isothermal stress-strain curves in glassy thermoplastics," *Proc. R. Soc. Lond. A*, vol. 302, no. 1471, pp. 453–472, 1968.

[185] G. Gurevich and P. Kobeko, "A study of polymers. iii. technique of mechanical tests of vulcanizates of rubber and plastics," *Rubber Chemistry and Technology*, vol. 13, no. 4, pp. 904–917, 1940.

[186] R. Haward, "The extension and rupture of cellulose acetate and celluloid," *Transactions of the Faraday Society*, vol. 38, pp. 394–403, 1942.

[187] E. Hoff, "Some mechanical properties of a commercial polymethyl methacrylate," *Journal of Applied Chemistry*, vol. 2, no. 8, pp. 441–448, 1952.

[188] R. Haward, B. Murphy, and E. White, "Relationship between compressive yield and tensile behavior in glassy thermoplastics," *Journal of Polymer Science Part A-2: Polymer Physics*, vol. 9, no. 5, pp. 801–814, 1971.

[189] M. C. Wang and E. Guth, "Statistical theory of networks of non-gaussian flexible chains," *The Journal of Chemical Physics*, vol. 20, no. 7, pp. 1144–1157, 1952.

[190] E. M. Arruda and M. C. Boyce, "A three-dimensional constitutive model for the large

stretch behavior of rubber elastic materials," *Journal of the Mechanics and Physics of Solids*, vol. 41, no. 2, pp. 389–412, 1993.

[191] R. S. Hoy and M. O. Robbins, "Strain hardening of polymer glasses: effect of entanglement density, temperature, and rate," *Journal of Polymer Science Part B: Polymer Physics*, vol. 44, no. 24, pp. 3487–3500, 2006.

[192] M. C. Boyce, D. M. Parks, and A. S. Argon, "Large inelastic deformation of glassy polymers. part i: rate dependent constitutive model," *Mechanics of Materials*, vol. 7, no. 1, pp. 15–33, 1988.

[193] M. Boyce and E. Arruda, "An experimental and anaiytical investigation of the large strain compressive and tensile response of glassy polymers," *Polymer Engineering & Science*, vol. 30, no. 20, pp. 1288–1298, 1990.

[194] H. M. James and E. Guth, "Theory of the elastic properties of rubber," *The Journal of Chemical Physics*, vol. 11, no. 10, pp. 455–481, 1943.

[195] P. Wu and E. Van Der Giessen, "On improved network models for rubber elasticity and their applications to orientation hardening in glassy polymers," *Journal of the Mechanics and Physics of Solids*, vol. 41, no. 3, pp. 427–456, 1993.

[196] R. Haward, "The application of a gauss-eyring model to predict the behavior of thermoplastics in tensile experiments," *Journal of Polymer Science Part B: Polymer Physics*, vol. 33, no. 10, pp. 1481–1494, 1995.

[197] C. G'sell, S. Boni, and S. Shrivastava, "Application of the plane simple shear test for determination of the plastic behaviour of solid polymers at large strains," *Journal of Materials Science*, vol. 18, no. 3, pp. 903–918, 1983.

[198] R. Haward, "Strain hardening of thermoplastics," *Macromolecules*, vol. 26, no. 22, pp. 5860–5869, 1993.

[199] A. Cross and R. Haward, "Orientation hardening of pvc," *Polymer*, vol. 19, no. 6, pp. 677–682, 1978.

[200] C. G'sell and J. Jonas, "Yield and transient effects during the plastic deformation of solid polymers," *Journal of Materials Science*, vol. 16, no. 7, pp. 1956–1974, 1981.

[201] M. Boyce and R. Haward, "The post-yield deformation of glassy polymers," in *The physics of glassy polymers*. Springer, 1997, pp. 213–293.

[202] E. M. Arruda, "Characterization of the strain hardening response of amorphous polymers," Ph.D. dissertation, Massachusetts Institute of Technology, 1992.

[203] T. A. Tervoort, *Constitutive modelling of polymer glasses: finite, nonlinear viscoelastic behaviour of polycarbonate*. Eindhoven University of Technology Eindhoven, 1996.

[204] H. Van Melick, L. Govaert, and H. Meijer, "On the origin of strain hardening in glassy polymers," *Polymer*, vol. 44, no. 8, pp. 2493–2502, 2003.

[205] E. M. Arruda, M. C. Boyce, and R. Jayachandran, "Effects of strain rate, temperature and thermomechanical coupling on the finite strain deformation of glassy polymers," *Mechanics of Materials*, vol. 19, no. 2-3, pp. 193–212, 1995.

[206] L. Govaert and T. Tervoort, "Strain hardening of polycarbonate in the glassy state: influence of temperature and molecular weight," *Journal of Polymer Science Part B: Polymer Physics*, vol. 42, no. 11, pp. 2041–2049, 2004.

[207] T. Tervoort and L. Govaert, "Strain-hardening behavior of polycarbonate in the glassy state," *Journal of Rheology*, vol. 44, no. 6, pp. 1263–1277, 2000.

[208] D. Rittel, "An investigation of the heat generated during cyclic loading of two glassy polymers. part i: Experimental," *Mechanics of Materials*, vol. 32, no. 3, pp. 131–147, 2000.

[209] Z. Li and J. Lambros, "Strain rate effects on the thermomechanical behavior of polymers," *International Journal of Solids and Structures*, vol. 38, no. 20, pp. 3549–3562, 2001.

[210] C. Chui and M. C. Boyce, "Monte carlo modeling of amorphous polymer deformation: Evolution of stress with strain," *Macromolecules*, vol. 32, no. 11, pp. 3795–3808, 1999.

[211] R. S. Hoy and M. O. Robbins, "Strain hardening in polymer glasses: limitations of network models," *Physical review letters*, vol. 99, no. 11, p. 117801, 2007.

[212] J. Li, T. Mulder, B. Vorselaars, A. V. Lyulin, and M. Michels, "Monte carlo simulation of uniaxial tension of an amorphous polyethylene-like polymer glass," *Macromolecules*, vol. 39, no. 22, pp. 7774–7782, 2006.

[213] R. S. Hoy and M. O. Robbins, "Strain hardening of polymer glasses: Entanglements, energetics, and plasticity," *Physical Review E*, vol. 77, no. 3, p. 031801, 2008.

[214] K. Chen and K. S. Schweizer, "Suppressed segmental relaxation as the origin of strain hardening in polymer glasses," *Physical review letters*, vol. 102, no. 3, p. 038301, 2009.

[215] M. Wendlandt, T. A. Tervoort, and U. W. Suter, "Non-linear, rate-dependent strain-hardening behavior of polymer glasses," *Polymer*, vol. 46, no. 25, pp. 11 786–11 797, 2005.

[216] C. Buckley, "In book of abstracts: 13th international conference on deformation," *Yield and Fracture of Polymers, Kerkrade, The Netherlands, MaTe, Eindhoven*, pp. 57–60, 2006.

[217] M. Wendlandt, T. A. Tervoort, and U. W. Suter, "Strain-hardening modulus of cross-linked glassy poly (methyl methacrylate)," *Journal of Polymer Science Part B: Polymer Physics*, vol. 48, no. 13, pp. 1464–1472, 2010.

[218] H. Li and C. Buckley, "Necking in glassy polymers: effects of intrinsic anisotropy

and structural evolution kinetics in their viscoplastic flow," *International Journal of Plasticity*, vol. 26, no. 12, pp. 1726–1745, 2010.

[219] L. Anand, N. M. Ames, V. Srivastava, and S. A. Chester, "A thermo-mechanically coupled theory for large deformations of amorphous polymers. part i: Formulation," *International Journal of Plasticity*, vol. 25, no. 8, pp. 1474–1494, 2009.

[220] V. Srivastava, S. A. Chester, N. M. Ames, and L. Anand, "A thermo-mechanically-coupled large-deformation theory for amorphous polymers in a temperature range which spans their glass transition," *International Journal of Plasticity*, vol. 26, no. 8, pp. 1138–1182, 2010.

[221] R. B. Dupaix and M. C. Boyce, "Constitutive modeling of the finite strain behavior of amorphous polymers in and above the glass transition," *Mechanics of Materials*, vol. 39, no. 1, pp. 39–52, 2007.

[222] P. Hempel, *Constitutive modeling of amorphous thermoplastic polymers with special emphasis on manufacturing processes*. KIT Scientific Publishing, 2016, vol. 2.

[223] R. Ball, M. Doi, S. Edwards, and M. Warner, "Elasticity of entangled networks," *Polymer*, vol. 22, no. 8, pp. 1010–1018, 1981.

[224] S. Edwards and T. Vilgis, "The effect of entanglements in rubber elasticity," *Polymer*, vol. 27, no. 4, pp. 483–492, 1986.

[225] J. Bergström and M. Boyce, "Constitutive modeling of the large strain time-dependent behavior of elastomers," *Journal of the Mechanics and Physics of Solids*, vol. 46, no. 5, pp. 931–954, 1998.

[226] P. Dooling, C. Buckley, S. Rostami, and N. Zahlan, "Hot-drawing of poly (methyl methacrylate) and simulation using a glass–rubber constitutive model," *Polymer*, vol. 43, no. 8, pp. 2451–2465, 2002.

[227] R. Xiao and T. D. Nguyen, "A thermodynamic modeling approach for dynamic softening in glassy amorphous polymers," *Extreme Mechanics Letters*, vol. 8, pp. 70–77, 2016.

[228] T. M. Nieuwenhuizen, "Thermodynamics of the glassy state: effective temperature as an additional system parameter," *Physical review letters*, vol. 80, no. 25, p. 5580, 1998.

[229] T. D. Nguyen, C. M. Yakacki, P. D. Brahmbhatt, and M. L. Chambers, "Modeling the relaxation mechanisms of amorphous shape memory polymers," *Advanced Materials*, vol. 22, no. 31, pp. 3411–3423, 2010.

[230] R. Xiao, J. Choi, N. Lakhera, C. M. Yakacki, C. P. Frick, and T. D. Nguyen, "Modeling the glass transition of amorphous networks for shape-memory behavior," *Journal of the Mechanics and Physics of Solids*, vol. 61, no. 7, pp. 1612–1635, 2013.

BIBLIOGRAPHY

[231] J. D. Ferry and J. D. Ferry, *Viscoelastic properties of polymers*. John Wiley & Sons, 1980.

[232] K. Kamrin and E. Bouchbinder, "Two-temperature continuum thermomechanics of deforming amorphous solids," *Journal of the Mechanics and Physics of Solids*, vol. 73, pp. 269–288, 2014.

[233] B. D. Coleman and M. E. Gurtin, "Thermodynamics with internal state variables," *The Journal of Chemical Physics*, vol. 47, no. 2, pp. 597–613, 1967.

[234] R. Xiao, G. Ghazaryan, T. A. Tervoort, and T. D. Nguyen, "Modeling energy storage and structural evolution during finite viscoplastic deformation of glassy polymers," *Physical Review E*, vol. 95, no. 6, p. 063001, 2017.

[235] A. Dawson, M. Rides, C. Allen, and J. Urquhart, "Polymer–mould interface heat transfer coefficient measurements for polymer processing," *Polymer Testing*, vol. 27, no. 5, pp. 555–565, 2008.

[236] P. Haupt, A. Lion, and E. Backhaus, "On the dynamic behaviour of polymers under finite strains: constitutive modelling and identification of parameters," *International Journal of Solids and Structures*, vol. 37, no. 26, pp. 3633–3646, 2000.

[237] C. Lindsey and G. Patterson, "Detailed comparison of the williams–watts and cole–davidson functions," *The Journal of chemical physics*, vol. 73, no. 7, pp. 3348–3357, 1980.

[238] T. Ge and M. O. Robbins, "Anisotropic plasticity and chain orientation in polymer glasses," *Journal of Polymer Science Part B: Polymer Physics*, vol. 48, no. 13, pp. 1473–1482, 2010.

[239] S. Lu and K. Pister, "Decomposition of deformation and representation of the free energy function for isotropic thermoelastic solids," *International Journal of Solids and Structures*, vol. 11, no. 7-8, pp. 927–934, 1975.

[240] A. Lion, "On the large deformation behaviour of reinforced rubber at different temperatures," *Journal of the Mechanics and Physics of Solids*, vol. 45, no. 11-12, pp. 1805–1834, 1997.

[241] M. C. Boyce, G. G. Wever, and D. M. Parks, "On the kinematics of finite strain plasticity," MASSACHUSETTS INST OF TECH CAMBRIDGE, Tech. Rep., 1989.

Vita

Jingkai Guo was born in Laiwu, Shandong Province, China in 1990. He received a B.S. in Theoretical and Applied Mechanics from Fudan University in June, 2012. He joined Professor Nguyen's lab at Johns Hopkins University in September 2012 and obtained a M.S.E in Mechanical Engineering in 2014. His research in Hopkins focused on the modeling of soft active materials including dielectric elastomers, hydrogels and shape memory polymers, which has been published in Journal of the Applied Mechanics and Extreme Mechanics Letters et al. His research interest lies in soft active materials and solid mechanics.